Administration Guide

/ Directory Services 6.5

Latest update: 6.5.6

Mark Craig
Nemanja Lukić
Ludovic Poitou
Chris Ridd

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

Abstract

Guide to configuring and using ForgeRock® Directory Services features.

Table of Contents

Preface vii
1. Understanding Directory Services 1
    How Directories and LDAP Evolved 2
    About Data In LDAP Directories 2
    About LDAP Client and Server Communication 6
    About LDAP Controls and Extensions 7
    About Indexes 8
    About LDAP Schema 8
    About Access Control 8
    About Replication 9
    About DSMLv2 9
    About RESTful Access to Directory Services 10
    About Building Directory Services 11
2. Administration Interfaces and Tools 12
    Server Command-Line Tools 12
    How Command-Line Tools Trust Server Certificates 18
    Using Configuration Property Value Substitution 19
3. Managing Server Processes 26
    Starting a Server 26
    Stopping a Server 27
    Restarting a Server 29
    Understanding Tasks 30
    Server Recovery 31
4. Managing Directory Data 32
    Generating Test Data 32
    Importing and Exporting Data 33
    Other Tools For Working With LDIF Data 35
    About Database Backends 38
    Creating a New Database Backend 39
    Splitting Data Across Multiple Backends 40
    Encrypting Directory Data 42
    Setting Disk Space Thresholds For Database Backends 47
    Automating Entry Expiration and Deletion 48
    Updating an Existing Backend to Add a New Base DN 49
    Deleting a Database Backend 50
5. Configuring Connection Handlers 51
    LDAP Client Access 51
    Preparing For Secure Communications 52
    LDAP Client Access With Transport Layer Security 66
    LDAP Client Access Over SSL 67
    Restricting Client Access 69
    TLS Protocols and Cipher Suites 71
    Client Certificate Validation and the Directory 76
    RESTful Client Access Over HTTP 77

DSML Client Access 92
JMX Client Access 94
LDIF File Access 96
SNMP Access 97
6. Configuring Privileges and Access Control 98
About Privileges 99
Configuring Privileges 101
About ACIs 105
Configuring ACIs 115
About Global Access Control Policies 122
Configuring Global Access Control Policies 123
Viewing Effective Rights 126
7. Indexing Attribute Values 130
About Indexes 130
What To Index 132
Index Types and Their Functions 140
Configuring and Rebuilding Indexes 145
Verifying Indexes 160
Default Indexes 160
8. Managing Data Replication 162
About Replication 162
Configuring Replication Settings 170
Change Notification For Your Applications 191
Recovering From User Error 201
Resolving Replication Conflicts 203
9. Backing Up and Restoring Data 206
Backing Up Directory Data 206
Restoring Directory Data From Backup 209
Backing Up and Restoring Configuration Files 212
10. Configuring Password Policy 214
About DS Password Policies 214
Configuring Password Policies 219
Assigning Password Policies 228
Configuring Password Generation 235
Configuring Password Storage 236
Configuring Password Validation 243
Sample Password Policies 245
11. Implementing Account Lockout and Notification 250
Configuring Account Lockout 250
Managing Accounts Manually 252
Managing Account Status Notification 253
12. Setting Resource Limits 257
Limiting Search Resources 257
Limiting and Restricting Client Connections 261
Limiting Idle Time 262
Limiting Maximum Request Size 263
Resource Limits and Proxied Authorization 263

13. Implementing Attribute Value Uniqueness 265
14. Managing Schema 272
About Directory Schema 272
Updating Directory Schema 275
Working With JSON 278
Relaxing Schema Checking to Import Legacy Data 281
Standard Schema Included With DS Servers 282
15. Configuring REST APIs 285
Adding a REST to LDAP Mapping for a Custom Object 285
Adding Subentry Password Policies 289
Mapping JSON Profiles to LDAP 294
Working With REST API Documentation 309
16. Configuring LDAP Proxy Services 313
About Proxy Services 313
Connecting to Remote Directory Servers 314
Routing Requests to Remote Directory Servers 317
Choosing Load Balancing Settings 319
Managing Schema Definitions 322
Configuring a Proxy Backend 322
Understanding How Failures Are Handled 325
Applying Access Control and Resource Limits for Proxy Services 326
Deploying Proxy Services for High Availability 327
Deploying a Single Point of Directory Access 327
17. Configuring Pass-Through Authentication 329
About Pass-Through Authentication 329
Setting Up Pass-Through Authentication 330
Assigning Pass-Through Authentication Policies 335
18. Samba Password Synchronization 338
19. Monitoring, Logging, and Alerts 341
HTTP-Based Monitoring 341
LDAP-Based Monitoring 344
SNMP-Based Monitoring 347
JMX-Based Monitoring 349
Server Operation and Tasks 350
Monitoring With Graphite 352
Server Logs 352
Alert Notifications 387
20. Tuning Servers For Performance 392
Defining Performance Requirements and Constraints 392
Testing Performance 394
Tweaking DS Performance 395
21. Securing and Hardening Servers 406
Setting Up a System Account for a Server 406
Using Java Security Updates 407
Only Enable Necessary Services 408
Configure Logging Appropriately 408
Limit Use of the cn=Directory Manager Account 408

Reconsider Default Global Access Control 409
Protect Network Connections 413
Use Appropriate Password Storage and Password Policies 414
Protect DS Server Files 416
22. Changing Server Certificates 418
23. Moving Servers 425
Overview 425
Before You Move 425
Moving a Server 427
24. Troubleshooting Server Problems 429
Identifying the Problem 429
Troubleshooting Installation and Upgrade 430
Resetting Administrator Passwords 431
Enabling Debug Logging 433
Preventing Access While Fixing Issues 434
Troubleshooting LDIF Import 436
Troubleshooting TLS/SSL Connections 436
Troubleshooting Client Operations 442
Troubleshooting Replication 444
Asking For Help 445
A. On Using a Load Balancer 446
The Problem With Load Balancers 446
Recommendations for Load Balancing 447
B. Getting Support 449

Preface

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.

This guide shows you how to configure, maintain, and troubleshoot Directory Services software. ForgeRock Directory Services allow applications to access directory data:

- Over Lightweight Directory Access Protocol (LDAP)
- Using Directory Services Markup Language (DSML)
- Over Hypertext Transfer Protocol (HTTP) by using HTTP methods in the Representational State Transfer (REST) style

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

- Use Directory Services administration tools
- Manage Directory Services processes
- Import, export, backup, and restore directory data
- Configure Directory Services connection handlers for all supported protocols
- Configure administrative privileges and fine-grained access control
- Index directory data, manage schemas for directory data, and enforce uniqueness of directory data attribute values
- Configure data replication
- Implement password policies, pass-through authentication to another directory, password synchronization with Samba, account lockout, and account status notification
- Set resource limits to prevent unfair use of server resources
- Monitor servers through logs and alerts and over JMX
- Tune servers for best performance

- Secure server deployments
- Change server key pairs and public key certificates
- Move a server to a different system
- Troubleshoot server issues

Chapter 1

Understanding Directory Services

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

- Why directory services exist and what they do well
- How data is arranged in directories that support Lightweight Directory Access Protocol (LDAP)
- How clients and servers communicate in LDAP
- What operations are standard according to LDAP and how standard extensions to the protocol work
- Why directory servers index directory data
- What LDAP schemas are for
- What LDAP directories provide to control access to directory data
- Why LDAP directory data is replicated and what replication does
- What Directory Services Markup Language (DSML) is for
- How HTTP applications can access directory data in the Representation State Transfer (REST) style

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

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

ForgeRock Directory Services are based on the Lightweight Directory Access Protocol (LDAP). Much of this chapter serves therefore as an introduction to LDAP. ForgeRock Directory Services also provide RESTful access to directory data, yet, as directory administrator, you will find it useful to

understand the underlying model even if most users are accessing the directory over HTTP rather than LDAP.

How Directories and LDAP Evolved

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

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

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

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

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

The DS code base began in the mid-2000s, when engineers solving the update performance issue decided 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.

About Data In LDAP Directories

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

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

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

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

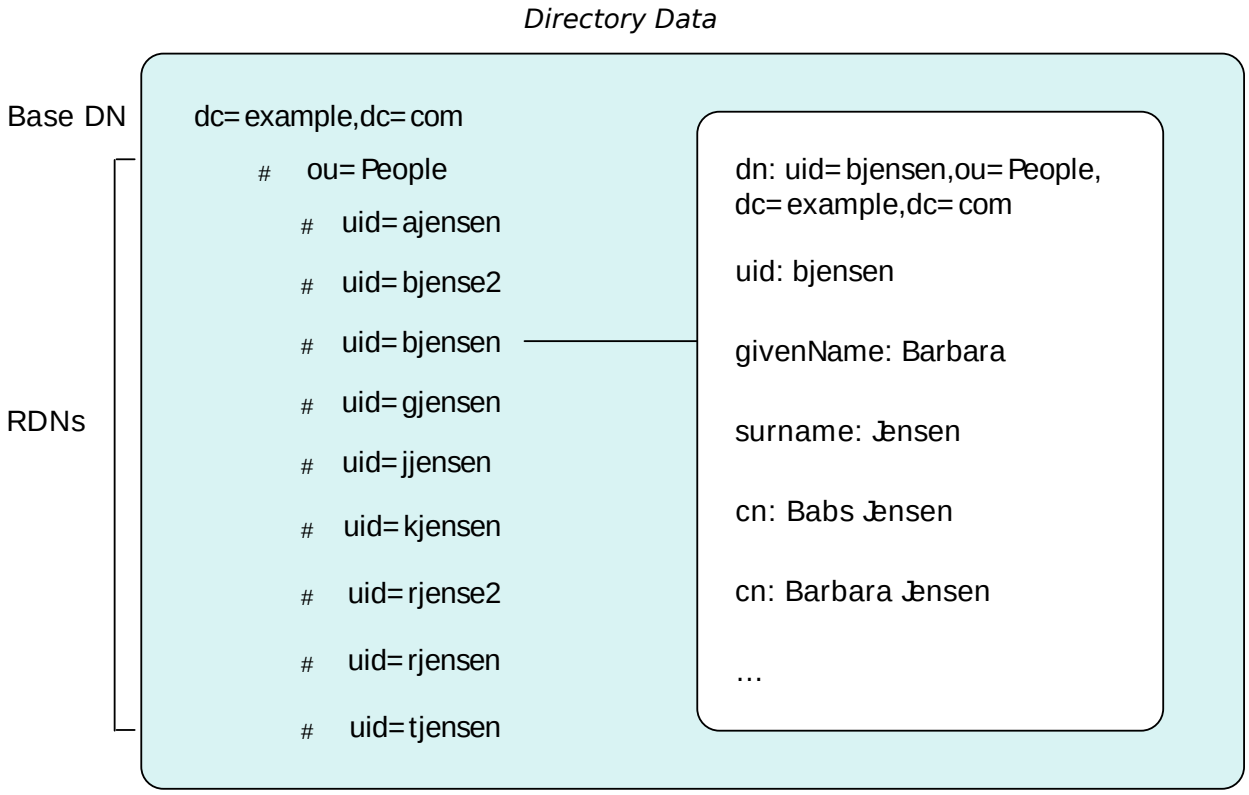
The entry also has a unique identifier, shown at the top of the entry, `dn: uid=bjensen,ou=People,dc=example,dc=com`. DN is an acronym for distinguished name. No two entries in the directory have the same distinguished name. Yet, DNs are typically composed of case-insensitive attributes.

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

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

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

"Directory Data" shows the hierarchy.



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

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

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

About LDAP Client and Server Communication

In some client server communication, like web browsing, a connection is set up and then torn down for each client request to the server. LDAP has a different model. In LDAP the client application connects to the server and authenticates, then requests any number of operations, perhaps processing results in between requests, and finally disconnects when done.

The standard operations are as follows:

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

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

- Search (lookup). After binding, the client can request that the server return entries based on an LDAP filter, which is an expression that the server uses to find entries that match the request, and a base DN under which to search. For example, to look up all entries for people with 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 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.

For practical examples showing how to perform the key operations using the command-line tools delivered with DS servers, read "*Performing LDAP Operations*" in the *Developer's Guide*.

About LDAP Controls and Extensions

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

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

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, thus 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 "*LDAP Controls*" in the *Reference*.

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, but then starts Transport Layer Security negotiations to protect communications.

For the list of supported LDAP extended operations, see "*LDAP Extended Operations*" in the *Reference*.

About Indexes

As mentioned early in this chapter, 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 "*Indexing Attribute Values*".

About LDAP Schema

Some databases are designed to hold huge amounts of data for a particular application. Although such databases might support multiple applications, how their data is organized depends a lot on the particular 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, many current directory administrators do not even know how many applications use their services. The shared, centralized nature of directory services fosters interoperability in practice, and has helped directory services be successful in the long term.

Part of what makes this possible is the shared model of directory user information, and in particular the LDAP schema. LDAP schema defines what the directory can contain. This means that directory entries are not arbitrary data, but instead 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 also 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 also without ever stopping your directory service.

For a closer look, see "*Managing Schema*".

About Access Control

In addition to directory schema, another feature of directory services that enables sharing is 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 what strength of encryption is needed in order to perform a particular operation.

As ACIs are stored on entries in the directory, you can furthermore 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 "*Configuring Privileges and Access Control*".

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

Another aspect of replication to keep in mind is that it is "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 right away. 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, either make sure their application always gets sent to the same server, or else ask that they adapt their application to work in a more realistic manner.

To get started with replication, see "*Managing Data Replication*".

About 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 "DSML Client Access".

About RESTful Access to Directory Services

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 and their 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 use should be the exception rather than the rule.) Furthermore, JSON resources can hold arrays, ordered collections that can contain duplicates, whereas 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.

For examples showing how to use RESTful access, see "*Performing RESTful Operations*" in the *Developer's Guide*.

About Building Directory Services

This chapter is meant to serve as an introduction, and so does not even cover everything in this guide, let alone everything you might want to know about directory services.

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. Read "*Tuning Servers For Performance*" for a look at how to meet the service levels that directory clients expect.

Chapter 2

Administration Interfaces and Tools

This chapter covers server administration tools. In this chapter you will learn to:

- Find and run command-line tools
- Understand how command-line tools trust server certificates
- Use expressions in the server configuration file

At a protocol level, administration tools and interfaces connect to servers through a different network port than that used to listen for traffic from other client applications.

This chapter takes a quick look at the tools for managing directory services.

Server Command-Line Tools

This section covers the tools installed with server software.

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

Paths To Administration Tools

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

You find the installation and upgrade tools, **setup**, and **upgrade**, in the parent directory of the other tools, as these tools are not used for everyday administration. For example, if the path to most tools is `/path/to/opendj/bin` you can find these tools in `/path/to/opendj`. For instructions on how to use the installation and upgrade tools, see the Installation Guide.

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

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

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

Tools and Server Constraints

Commands	Constraints
backendstat **create-rc-script** **encode-password** **setup** **start-ds** **supportextract** **upgrade** **windows-service**	These commands must be used with the local DS server in the same installation as the tools. These commands are not useful with non-DS servers.
dsconfig **export-ldif** **import-ldif** **manage-account** **manage-tasks** **rebuild-index** **restore** **status** **stop-ds** **verify-index**	These commands must be used with DS servers having the same version as the command. These commands are not useful with non-DS servers.
dsreplication	• With one exception, this command can be used with current and previous DS server versions. The one exception is the **dsreplication reset-change-number** subcommand, which requires DS server version 3.0.0 or later. • This command does not support configuration expressions described in "Using Configuration Property Value Substitution". When setting up and initializing replication in a DevOps environment, use the **dsreplication** command before substituting configuration values with expressions. • This command is not useful with other types of directory servers.
makeldif	This command depends on template files. The template files can make use of configuration files installed with DS servers under `config/MakeLDIF/`. The LDIF output can be used with any directory server.
base64 **ldapcompare** **ldapdelete** **ldapmodify** **ldappasswordmodify**	These commands can be used independently of DS servers, and so are not tied to a specific version.

Commands	Constraints
ldapsearch **ldifdiff** **ldifmodify** **ldifsearch**	

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

addrate

Measure add and delete throughput and response time.

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

authrate

Measure bind throughput and response time.

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

backendstat

Debug databases for pluggable backends.

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

backup

Back up or schedule backup of directory data.

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

base64

Encode and decode data in base64 format.

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

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

changelogstat

Debug file-based changelog databases.

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

create-rc-script (UNIX)

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

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

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

dsconfig

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

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

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

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

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

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

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

dsreplication

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

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

encode-password

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

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

export-ldif

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

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

import-ldif

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

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

ldapcompare

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

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

ldapdelete

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

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

ldapmodify

Modify the specified attribute values for the specified entries.

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

ldappasswordmodify

Modify user passwords.

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

ldapsearch

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

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

ldifdiff

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

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

ldifmodify

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

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

ldifsearch

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

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

makeldif

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

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

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

manage-account

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

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

manage-tasks

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

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

modrate

Measure modification throughput and response time.

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

rebuild-index

Rebuild an index stored in an indexed backend.

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

restore

Restore data from backup.

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

searchrate

Measure search throughput and response time.

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

start-ds

Start one DS server.

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

status

Display information about the server.

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

stop-ds

Stop one DS server.

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

supportextract

Collect troubleshooting information for technical support purposes.

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

verify-index

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

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

windows-service (Windows)

Register and manage one DS server as a Windows Service.

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

How Command-Line Tools Trust Server Certificates

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

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

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

Put simply, a tool can automatically trust the server certificate if the tool's truststore contains the signing certificate. "Command-Line Tools and Truststores" indicates where to find the truststore. The signing certificate could be a CA certificate, or the server certificate itself if the certificate was self-signed.

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

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

Command-Line Tools and Truststores

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

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

Using Configuration Property Value Substitution

Property value substitution enables you to achieve the following:

- Define a configuration that is specific to a single instance, for example, setting the location of the keystore on a particular host.
- Define a configuration whose parameters vary between different environments, for example, the hostnames and passwords for test, development, and production environments.
- Disable certain capabilities on specific servers. For example, you might want to disable a database backend and its replication agreement for one set of replicas while enabling it on another set of replicas. This makes it possible to use the same configuration for environments using different data sets.

Property value substitution uses *configuration expressions* to introduce variables into the server configuration. You set configuration expressions as the values of configuration properties. The effective property values can be evaluated in a number of ways. For more information about property evaluation, see "Expression Evaluation and Order of Precedence".

Note

DS servers only resolve configuration expressions in the `config/config.ldif` file on LDAP attributes whose names start with `ds-cfg-*`. These correspond to configuration properties listed in "*Properties*" in the *Configuration Reference*.

DS servers do not resolve configuration expressions anywhere else.

DS servers resolve expressions at startup to determine the configuration. DS commands that read the configuration in offline mode also resolve expressions at startup. When you use expressions in the

configuration, you must make their values available as described below before starting the server and also when running such commands.

Configuration expressions share their syntax and underlying implementation with other ForgeRock Identity Platform software. Configuration expressions have the following characteristics:

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

  For example, the following expression sets the default listen port value to 1389: `&{listen.port|1389}`.
- A configuration property can include a mix of static values and expressions.

  For example, suppose `hostname` is set to `directory`. Then `&{hostname}.example.com` evaluates to `directory.example.com`.
- You can define *nested* properties (that is a property definition within another property definition).

  For example, suppose `listen.port` is set to `&{port.prefix}389`, and `port.prefix` is set to `2`. Then `&{listen.port}` evaluates to `2389`.
- You can read the value of an expression token from a file.

  For example, if the cleartext password is stored in `/path/to/password.txt`, the following expression resolves to the cleartext password: `&{file:/path/to/password.txt}`.

  You specify the file either by its absolute path, or by a path relative to the DS instance directory. In other words, if the DS instance directory is `/path/to/opendj`, then `/path/to/opendj/config/keystore.pin` and `config/keystore.pin` reference the same file.

DS servers define the following expression tokens by default. You can use these in expressions without explicitly setting their values beforehand:

`ds.instance.dir`

The file system directory holding the instance files required to run an instance of a server.

By default, the files are co-located with the product tools, libraries, and configuration files. You can change the location by using the `setup --instancePath` option.

This evaluates to a directory such as `/path/to/my-instance`.

`ds.install.dir`

The file system directory where the server files are installed.

This evaluates to a directory such as `/path/to/opendj`.

Expression Evaluation and Order of Precedence

You must define expression values before starting the DS server that uses them. When evaluated, an expression must return the appropriate type for the configuration property. For example, the `listen-port` property takes an integer. If you set it using an expression, the result of the evaluated expression must be an integer. If the type is wrong, the server fails to start due to a syntax error.

If the expression cannot be resolved, and there is no default value in the configuration expression, DS also fails to start.

Expression resolvers evaluate expression tokens to literal values.

Expression resolvers get values from the following sources:

1. **Environment variables**

   You set an environment variable to hold the value.

   For example: `export LISTEN_PORT=1389`

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

   - Uppercase characters are lower cased.
   - Underscores, `_`, are replaced with `.` characters.

   In other words, the value of `LISTEN_PORT` replaces `&{listen.port}` in the server configuration.

2. **Java system properties**

   You set a Java system property to hold the value.

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

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

   For example: `export OPENDJ_JAVA_ARGS="-Dlisten.port=1389"`

3. **Expressions files (optional)**

   You set a key in a `.json` or `.properties` file to hold the value. This optional mechanism is set using the `DS_ENVCONFIG_DIRS` environment variable, or the `ds.envconfig.dirs` Java system property as described below.

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

   The following example expressions properties file sets the listen port:

```
listen.port=1389
```

JSON expression files can contain nested objects.

JSON field names map to expression tokens as follows:

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

The following example JSON expressions file sets the listen address and listen port:

```
{
  "listen": {
    "address": "192.168.0.10",
    "port": "1389"
  }
}
```

In other words, the value of the `listen/port` field replaces `&{listen.port}` in the server configuration.

In order to use expression files, set the environment variable, `DS_ENVCONFIG_DIRS`, or the Java system property, `ds.envconfig.dirs`, to a comma-separated list of the directories containing the expression files.

Note the following constraints when using expression files:

- Although DS browses the directories in the specified order, within a directory DS scans the files in a non-deterministic order.
- DS reads all files with `.json` and `.properties` extensions.
- DS does not scan subdirectories.
- Do not define the same configuration token more than once in a file, as you cannot know in advance which value will be used.
- You cannot define the same configuration token in more than one file in a single directory.

  If the same token occurs in more than one file in a single directory, an error occurs.
- If the same token occurs once in several files which are located in different directories, the first value that DS reads is used.

The preceding list reflects the order of precedence:

- Environment variables override system properties, default token settings, and settings in any expression files.
- System properties override default token settings, and any settings in expression files.
- If `DS_ENVCONFIG_DIRS` or `ds.envconfig.dirs` is set, then the server uses settings found in expression files.

- Default token settings (`ds.config.dir`, `ds.instance.dir`, `ds.install.dir`).

For an embedded DS server, it is possible to change the expression resolvers, in the server configuration. For details, see the DS server Java API.

Using Multivalued Expressions

A single expression token can evaluate to multiple property values. Such expressions are useful with multivalued properties.

For example, suppose you choose to set a connection handler's `ssl-cipher-suite` property. Instead of listing cipher suites individually, you use an `ssl.cipher.suites` token that takes multiple values. The following example commands set the token value in the environment, stop the server, use the expression in the LDAP connection handler configuration while the server is offline, and then start the server again:

```
$ export SSL_CIPHER_SUITES=\
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
$ stop-ds --quiet
$ dsconfig \
 set-connection-handler-prop \
 --offline \
 --handler-name LDAP \
 --add ssl-protocol:TLSv1.2 \
 --add ssl-cipher-suite:'&{ssl.cipher.suites}' \
 --no-prompt
$ start-ds --quiet
```

Multiple values are separated by commas in environment variables, system properties, and properties files. They are formatted as arrays in JSON files.

You can set the value of the `ssl.cipher.suites` token in the following ways:

Environment Variable

```
export SSL_CIPHER_SUITES=\
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
```

System Property

```
export OPENDJ_JAVA_ARGS="-Dssl.cipher.suites=\
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\
TLS_EMPTY_RENEGOTIATION_INFO_SCSV"
```

Properties File

```
ssl.cipher.suites=\
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,\
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
```

JSON Expressions File

```
{
  "ssl.cipher.suites": [
    "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
    "TLS_EMPTY_RENEGOTIATION_INFO_SCSV"
  ]
}
```

Alternative JSON expressions file that sets `ssl.protocol` as well:

```
{
 "ssl": {
   "protocol": "TLSv1.2",
   "cipher.suites": [
     "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
     "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
     "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
     "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
     "TLS_EMPTY_RENEGOTIATION_INFO_SCSV"
   ]
 }
}
```

In order to fully use the settings in this file, you would have to change the example to include the additional expression: `--add ssl-protocol:'&{ssl.protocol}'`.

When the server evaluates `&{ssl.cipher.suites}`, the result is the following property values:

```
ssl-cipher-suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
ssl-cipher-suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
ssl-cipher-suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
ssl-cipher-suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
ssl-cipher-suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV
```

Debugging Expressions

You can debug configuration expressions by following the instructions in "Enabling Debug Logging". Create a debug target for `org.forgerock.config.resolvers`. The following example demonstrates the process:

```
$ dsconfig \
 create-debug-target \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --type generic \
 --target-name org.forgerock.config.resolvers \
 --set enabled:true \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
$ stop-ds --restart --quiet
```

When the server starts, it logs debugging messages for configuration expressions. Do not leave debug logging enabled in production systems.

Chapter 3

Managing Server Processes

This chapter covers starting and stopping DS servers. In this chapter you will learn to:

- Start, restart, and stop servers using the command-line tools or system service integration on Linux and Windows systems
- Understand server tasks and how to schedule them
- Understand how to recognize that a server is recovering from a crash or abrupt shutdown
- Configure whether a directory server loads data into cache at server startup before accepting client connections

Starting a Server

Use one of the following techniques:

- Use the **start-ds** command, described in "*start-ds — start OpenDJ server*" in the *Reference*:

  ```
  $ start-ds
  ```

  Alternatively, you can specify the `--no-detach` option to start the server in the foreground.

- (Linux) If the DS server was installed from a .deb or .rpm package, then service management scripts were created at setup time.

  Use the **service opendj start** command:

  ```
  centos# service opendj start
  Starting opendj (via systemctl):                           [  OK  ]
  ```

  ```
  ubuntu$ sudo service opendj start
  $Starting opendj: > SUCCESS.
  ```

- (UNIX) Create an RC script by using the **create-rc-script** command, described in "*create-rc-script — script to manage OpenDJ as a service on UNIX*" in the *Reference*, and then use the script to start the server.

Unless you run DS servers on Linux as root, use the `--userName` *`userName`* option to specify the user who installed the server:

```
$ sudo create-rc-script \
 --outputFile /etc/init.d/opendj \
 --userName opendj

$ sudo /etc/init.d/opendj start
```

For example, if you run the DS server on Linux as root, you can use the RC script to start the server at system boot, and stop the server at system shutdown:

```
$ sudo update-rc.d opendj defaults
update-rc.d: warning: /etc/init.d/opendj missing LSB information
update-rc.d: see <http://wiki.debian.org/LSBInitScripts>
 Adding system startup for /etc/init.d/opendj ...
   /etc/rc0.d/K20opendj -> ../init.d/opendj
   /etc/rc1.d/K20opendj -> ../init.d/opendj
   /etc/rc6.d/K20opendj -> ../init.d/opendj
   /etc/rc2.d/S20opendj -> ../init.d/opendj
   /etc/rc3.d/S20opendj -> ../init.d/opendj
   /etc/rc4.d/S20opendj -> ../init.d/opendj
   /etc/rc5.d/S20opendj -> ../init.d/opendj
```

- (Windows) Register DS servers as Windows Services by using the **windows-service** command, described in "*windows-service — register DS as a Windows Service*" in the *Reference*. Manage the service through Windows administration tools. The following example registers the DS server as a Windows Service:

```
C:\path\to\opendj\bat> windows-service.bat --enableService
```

By default, DS servers save a compressed version of the server configuration used on successful startup. This ensures that the server provides a last known good configuration, which can be used as a reference or copied into the active configuration if the server fails to start with the current active configuration. It is possible, though not usually recommended, to turn this behavior off by changing the global server setting `save-config-on-successful-startup` to `false`.

Stopping a Server

Although DS servers are designed to recover from failure and disorderly shutdown, it is safer to shut the server down cleanly, because a clean shutdown reduces startup delays during which the server attempts to recover database backend state, and prevents situations where the server cannot recover automatically.

Follow these steps to shut down the DS server cleanly:

To Stop a Server Cleanly

1. (Optional) If you are stopping a replicated server *permanently*, for example, before decommissioning the underlying system or virtual machine, first remove the server from the replication topology:

```
$ dsreplication \
 unconfigure \
 --unconfigureAll \
 --port 4444 \
 --hostname opendj.example.com \
 --adminUID admin \
 --adminPassword password \
 --trustAll \
 --no-prompt
```

   This step unregisters the server from the replication topology, effectively removing its replication configuration from other servers. This step must be performed before you decommission the system, because the server must connect to its peers in the replication topology.

2. Before shutting down the system where the server is running, and before detaching any storage used for directory data, cleanly stop the server using one of the following techniques:

   - Use the **stop-ds** command, described in "*stop-ds — stop OpenDJ server*" in the *Reference*:

```
$ stop-ds
```

   - (Linux) If the DS server was installed from a .deb or .rpm package, then service management scripts were created at setup time.

     Use the **service opendj stop** command:

```
centos# service opendj stop
Stopping opendj (via systemctl):                           [  OK  ]
```

```
ubuntu$ sudo service opendj stop
$Stopping opendj: ... > SUCCESS.
```

   - (UNIX) Create an RC script, and then use the script to stop the server:

```
$ sudo create-rc-script \
 --outputFile /etc/init.d/opendj \
 --userName opendj

$ sudo /etc/init.d/opendj stop
```

- (Windows) Register the DS server as a Windows Service, and then manage the service through Windows administration tools:

```
C:\path\to\opendj\bat> windows-service.bat --enableService
```

Do not intentionally kill the DS server process unless the server is completely unresponsive.

When stopping cleanly, the server writes state information to database backends, and releases locks that it holds on database files.

Restarting a Server

Use one of the following techniques:

- Use the **stop-ds** command:

```
$ stop-ds --restart
```

- (Linux) If the DS server was installed from a .deb or .rpm package, then service management scripts were created at setup time.

  Use the **service opendj restart** command:

```
centos# service opendj restart
Restarting opendj (via systemctl):                         [  OK  ]
```

```
ubuntu$ sudo service opendj restart
$Stopping opendj: ... > SUCCESS.

$Starting opendj: > SUCCESS.
```

- (UNIX) Create an RC script, and then use the script to stop the server:

```
$ sudo create-rc-script \
 --outputFile /etc/init.d/opendj \
 --userName opendj

$ sudo /etc/init.d/opendj restart
```

- (Windows) Register the DS server as a Windows Service, and then manage the service through Windows administration tools:

```
C:\path\to\opendj\bat> windows-service.bat --enableService
```

Understanding Tasks

The following server administration commands can be run in online and offline mode. They invoke data-intensive operations, and so potentially take a long time to complete. The links below are to the reference documentation for each command:

- "*backup — back up directory data*" in the *Reference*

  "*export-ldif — export directory data in LDIF*" in the *Reference*

  "*import-ldif — import directory data from LDIF*" in the *Reference*

  "*restore — restore directory data backups*" in the *Reference*

  "*rebuild-index — rebuild index after configuration change*" in the *Reference*

When you run these commands in online mode, they run as *tasks* on the server. Server tasks are scheduled operations that can run one or more times as long as the server is up. For example, you can schedule the **backup** and **export-ldif** commands to run recurrently in order to back up server data on a regular basis. See the command's reference documentation for details.

You schedule a task as a directory administrator, sending the request to the administration port. You can therefore schedule a task on a remote server if you choose. When you schedule a task on a server, the command returns immediately, yet the task can start later, and might run for a long time before it completes. You can access tasks by using the "*manage-tasks — manage server administration tasks*" in the *Reference* command.

Although you can schedule a server task on a remote server, *the data for the task must be accessible to the server locally*. For example, when you schedule a backup task on a remote server, that server writes backup files to a file system on the remote server. Similarly, when you schedule a restore task on a remote server, that server restores backup files from a file system on the remote server.

The reference documentation describes the available options for each command:

- Configure email notification for success and failure

- Define alternatives on failure
- Start tasks immediately (`--start 0`)
- Schedule tasks to start at any time in the future

Server Recovery

DS servers tend to show resilience when restarting after a crash or after the server process is killed abruptly. After disorderly shutdown, the DS server might have to replay the last few entries in a transaction log. Generally, DS servers return to service quickly.

Database recovery messages are found in the database log file, such as `/path/to/opendj/db/userRoot/dj.log`.

The following example shows two example messages from the recovery log. The first message is written at the beginning of the recovery process. The second message is written at the end of the process:

```
[/path/to/opendj/db/userRoot]Recovery underway, found end of log
...
[/path/to/opendj/db/userRoot]Recovery finished: Recovery Info ...
```

Directory data cached in memory is lost during a crash. Loading directory data into memory when the server starts can take time. DS servers start accepting client requests before this process is complete.

Chapter 4

Managing Directory Data

This chapter covers management of LDAP Data Interchange Format (LDIF) data. In this chapter you will learn to:

- Generate test LDIF data
- Import and export LDIF data
- Perform searches and modifications on LDIF files with command-line tools
- Create and manage database backends to house directory data imported from LDIF
- Delete database backends

LDIF provides a mechanism for representing directory data in text format. LDIF data is typically used to initialize directory databases, but also may be used to move data between different directories that cannot replicate directly, or even as an alternative backup format.

Generating Test Data

When you install DS directory servers, you have the option of importing sample data that is generated during the installation. This procedure demonstrates how to generate LDIF by using the **makeldif** command, described in "*makeldif — generate test LDIF*" in the *Reference*.

To Generate Test LDIF Data

The **makeldif** command uses templates to provide sample data. Default templates are located in the `/path/to/opendj/config/MakeLDIF/` directory. The `example.template` file can be used to create a suffix with entries of the type `inetOrgPerson`.

1. Write a file to act as the template for your generated LDIF.

   The resulting test data template depends on what data you expect to encounter in production. Base your work on your knowledge of the production data, and on the sample template, `/path/to/opendj/config/MakeLDIF/example.template`, and associated data.

   See "*makeldif.template — template file for the makeldif command*" in the *Reference* for reference information about template files.

2. Create additional data files for the content in your template to be selected randomly from a file, rather than generated by an expression.

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

3. Decide whether you want to generate the same test data each time you run the **makeldif** command with your template.

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

4. Before generating a very large LDIF file, make sure you have enough space on disk.

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

   The following command demonstrates use of the example MakeLDIF template:

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

Importing and Exporting Data

The following procedures demonstrate how to use the **import-ldif** and **export-ldif** commands, described in "*import-ldif — import directory data from LDIF*" in the *Reference* and "*export-ldif — export directory data in LDIF*" in the *Reference*.

To Import LDIF Data

The most efficient method of importing LDIF data is to take the DS server offline. Alternatively, you can schedule a task to import the data while the server is online.

Note

Importing from LDIF overwrites all data in the target backend with entries from the LDIF data.

1. The following example imports `dc=example,dc=com` data into the `dsEvaluation` backend, overwriting existing data:

   - If you want to speed up the process—for example because you have millions of directory entries to import—first shut down the server, and then run the **import-ldif** command:

```
$ stop-ds --quiet
$ import-ldif \
 --offline \
 --backendID dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile generated.ldif
```

- If not, schedule a task to import the data while online:

```
$ start-ds --quiet
$ import-ldif \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backendID dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile generated.ldif \
 --trustAll
```

  The import task is scheduled through a secure connection to the administration port, by default `4444`. You can schedule the import task to start at a particular time using the `--start` option.

  The `--trustAll` option trusts all SSL certificates, such as a self-signed DS server certificate.

2. If the server is replicated with other servers, initialize replication again after the successful import.

   For details see "Initializing Replicas".

   Initializing replication overwrites data in the remote servers in the same way that import overwrites existing data with LDIF data.

   If the LDIF was exported from a server rather than generated using the **makeldif** command, it may contain pre-encoded passwords. By default, password policies do not allow you to use pre-encoded passwords, but you can change this behavior by changing the password policy's configuration property, `allow-pre-encoded-passwords`. Furthermore, the LDIF may include passwords encrypted using a reversible storage scheme, such as AES or Blowfish. As described in "Configuring Password Storage", in order to decrypt the passwords, the server must be a replica of the server that encrypted the passwords.

To Export LDIF Data

The following examples export `dc=example,dc=com` data from the `dsEvaluation` backend:

1. To expedite export, shut down the server and then use the **export-ldif** command:

```
$ stop-ds --quiet
$ export-ldif \
 --offline \
 --backendID dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile backup.ldif
```

2. To export the data while online, leave the server running and schedule a task:

```
$ export-ldif \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backendID dsEvaluation \
 --includeBranch dc=example,dc=com \
 --ldifFile backup.ldif \
 --start 0 \
 --trustAll
```

The `--start 0` option tells the directory server to start the export task immediately.

You can specify a time for the task to start using the format *yyyymmddHHMMSS*. For example, `20250101043000` specifies a start time of 4:30 AM on January 1, 2025.

If the server is not running at the time specified, it attempts to perform the task after it restarts.

Other Tools For Working With LDIF Data

This section demonstrates the command-line tools for working with LDIF:

- ldifdiff, demonstrated in "Comparing LDIF Files".
- ldifmodify, demonstrated in "Updating LDIF".
- ldifsearch, demonstrated in "Searching LDIF".

Searching LDIF

The **ldifsearch** command is to LDIF files what the **ldapsearch** command is to directory servers:

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

The *ldif-file* replaces the `--hostname` and `--port` options used to connect to an LDAP directory. Otherwise, the command syntax and LDIF output is familiar to **ldapsearch** users.

Updating LDIF

The **ldifmodify** command lets you apply changes to LDIF files, generating a new, changed version of the original file:

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


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

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

Comparing LDIF Files

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

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

The **ldifdiff** command reads both files into memory, and constructs tree maps to perform the comparison. The command is designed to work with small files and fragments, and can quickly run out of memory when calculating differences between large files.

Using Standard Input With the LDIF Tools

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

How bare dashes can be used after a double dash depends on the tool:

ldifdiff

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

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

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

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

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

ldifmodify

The bare dash can replace either the source LDIF file, or the changes LDIF file arguments.

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

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

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

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

ldifsearch

The bare dash can be used to take the source LDIF from standard input by using the following construction:

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

About Database Backends

DS directory server stores data in a *backend*. A backend is a private server repository that can be implemented in memory, as a file, or as an embedded database.

Database backends are designed to hold large amounts of user data. DS servers have tools for backing up and restoring database backends, as described in "*Backing Up and Restoring Data*". By default, a directory server stores your data in a database backend named `userRoot`, unless you use a setup profile.

When installing the server and importing user data, and when creating a database backend, you choose the backend type. DS directory servers use JE backends for local data.

The JE backend type is implemented using B-tree data structures. It stores data as key-value pairs, which is different from the relational model exposed to clients of relational databases.

Important

Do not compress, tamper with, or otherwise alter backend database files directly unless specifically instructed to do so by a qualified ForgeRock technical support engineer. External changes to backend database files can render them unusable by the server. By default, backend database files are located under the `/path/to/opendj/db` directory.

When backing up backend databases at the file system level rather than using the **backup** command, be aware that you may need to stop the server before running the backup procedure. A database backend performs internal cleanup operations that change the database log files even when there are no pending client or replication operations. An ongoing file system backup operation may therefore record database log files that are not in sync with each other. Such inconsistencies make it impossible for the backend database to recover after the database log files are restored. When the directory server is online during file system backup, successful recovery after restore can only be guaranteed if the backup operation took a fully atomic snapshot, capturing the state of all files at exactly the same time. If the recursive file system copy takes a true snapshot, you may perform the backup with the DS server online. Otherwise, if the file system copy is not a fully atomic snapshot, then you *must stop the DS server before performing the backup operation*.

A JE backend stores data on disk using append-only log files with names like *`number`*`.jdb`. The JE backend writes updates to the highest-numbered log file. The log files grow until they reach a specified size (default: 1 GB). When the current log file reaches the specified size, the JE backend creates a new log file.

To avoid an endless increase in database size on disk, JE backends clean their log files in the background. A cleaner thread copies active records to new log files. Log files that no longer contain active records are deleted.

By default, JE backends let the operating system potentially cache data for a period of time before flushing the data to disk. This setting trades full durability with higher disk I/O for good performance with lower disk I/O. With this setting, it is possible to lose the most recent updates that were not yet written to disk in the event of an underlying OS or hardware failure. You can modify this behavior by changing the advanced configuration settings for the JE backend. If necessary, you can change the advanced setting for durability, db-durability, using the **dsconfig set-backend-prop** command.

When a JE backend is opened, it recovers by recreating its B-tree structure from its log files. This is a normal process, one that allows the backend to recover after an orderly shutdown or after a crash.

Due to the cleanup processes, JE backends can be actively writing to disk even when there are no pending client or replication operations. To back up a server using a file system snapshot, you must *stop the server before taking the snapshot*.

In addition to the cleanup process, JE backends run checksum verification periodically on the database logs. If the verification process detects backend database corruption, then the server logs an error message and the backend is taken offline. In this case, restore the corrupted backend from backup so that it can be used again. By default, the verification runs every night at midnight local time. If necessary, you can change this behavior by adjusting the advanced settings, db-run-log-verifier, and db-log-verifier-schedule, using the **dsconfig set-backend-prop** command.

Creating a New Database Backend

You can create new backends using the **dsconfig create-backend** command, described in "create-backend" in the *Configuration Reference*. When you create a backend, choose the type of backend that fits your purpose.

The following example creates a database backend named `exampleOrgBackend`. The backend is of type `je`, which relies on a JE database for data storage and indexing:

```
$ dsconfig \
 create-backend \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name exampleOrgBackend \
 --type je \
 --set enabled:true \
 --set base-dn:dc=example,dc=org \
 --trustAll \
 --no-prompt
```

After creating the backend, you can view the settings as in the following example:

```
$ dsconfig \
 get-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name exampleOrgBackend \
 --trustAll \
 --no-prompt
Property                 : Value(s)
-------------------------:---------------------
backend-id               : exampleOrgBackend
base-dn                  : "dc=example,dc=org"
cipher-key-length        : 128
cipher-transformation    : AES/CBC/PKCS5Padding
compact-encoding         : true
confidentiality-enabled  : false
db-cache-percent         : 50
db-cache-size            : 0 b
db-directory             : db
enabled                  : true
writability-mode         : enabled
```

When you create a new backend using the **dsconfig** command, DS directory servers create the following indexes automatically:

`aci` presence
`ds-sync-conflict` equality
`ds-sync-hist` ordering
`entryUUID` equality
`objectClass` equality

You can create additional indexes as described in "Configuring and Rebuilding Indexes".

Splitting Data Across Multiple Backends

In some cases, such as directory services with subtree replication, you might choose to split directory data across multiple backends.

In the example in this section, the `exampleData` backend holds all data under `dc=example,dc=com` except `ou=People,dc=example,dc=com`. A separate `peopleData` backend holds data under `ou=People,dc=example,dc=com`. Replication for these backends is configured separately as described in "Subtree Replication".

The following example assumes you perform the steps when initially setting up the directory data. It uses the sample data from "Generating Test Data":

```
# Create a backend for the data not in the sub-suffix:
$ dsconfig \
 create-backend \
 --hostname opendj.example.com \
```

```
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name exampleData \
 --type je \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --trustAll \
 --no-prompt

# Create a backend for the data in the sub-suffix:
$ dsconfig \
 create-backend \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name peopleData \
 --type je \
 --set enabled:true \
 --set base-dn:ou=People,dc=example,dc=com \
 --trustAll \
 --no-prompt

# Import data not in the sub-suffix:
$ import-ldif \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backendID exampleData \
 --excludeBranch ou=People,dc=example,dc=com \
 --ldifFile generated.ldif \
 --trustAll

# Import data in the sub-suffix:
$ import-ldif \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backendID peopleData \
 --includeBranch ou=People,dc=example,dc=com \
 --ldifFile generated.ldif \
 --trustAll
```

If, unlike the example, you must split an existing backend, follow these steps:

1. Create the new backend.
2. Export the data for the backend.
3. Import the data for the sub-suffix into the new backend using the `--includeBranch` option.
4. Delete all data under the sub-suffix from the old backend.

   For an example, see "Delete: Removing a Subtree" in the *Developer's Guide*.

When you split an existing backend, you must handle the service interruption that results when you move the sub-suffix data out of the original backend. If there is not a maintenance window where you can bring the service down in order to move the data, consider alternatives for replaying changes applied between the time that you exported the data and the time that you retired the old sub-suffix.

Encrypting Directory Data

DS directory servers can encrypt directory data before storing it in a database backend on disk, keeping the data confidential until it is accessed by a directory client.

Data encryption is useful for at least the following cases:

Ensuring Confidentiality and Integrity

Encrypted directory data is confidential, remaining private until decrypted with a proper key.

Encryption ensures data integrity at the moment it is accessed. The DS directory service cannot decrypt corrupted data.

Protection on a Shared Infrastructure

When you deploy directory services on a shared infrastructure you relinquish full and sole control of directory data.

For example, if the DS directory server runs in the cloud, or in a data center with shared disks, the file system and disk management are not under your control.

Data confidentiality and encryption come with the following trade-offs:

Equality Indexes Limited to Equality Matching

When an equality index is configured without confidentiality, the values can be maintained in sorted order. A non-confidential, cleartext equality index can therefore be used for searches that require ordering and searches that match an initial substring.

An example of a search that requires ordering is a search with a filter `"(cn<=App)"`. The filter matches entries with `commonName` up to those starting with `App` (case-insensitive) in alphabetical order.

An example of a search that matches an initial substring is a search with a filter `"(cn=A*)"`. The filter matches entries having a `commonName` that starts with `a` (case-insensitive).

In an equality index with confidentiality enabled, the DS directory server no longer sorts cleartext values. As a result, you must accept that ordering and initial substring searches are unindexed.

Performance Impact

Encryption and decryption requires more processing than handling cleartext values.

Encrypted values also take up more space than cleartext values.

As explained in "Protect DS Server Files", DS directory servers do not encrypt directory data by default. This means that any user with system access to read directory files can potentially access directory data in cleartext.

You can verify what a system user could see with read access to backend database files by using the **backendstat dump-raw-db** command. The **backendstat** subcommands **list-raw-dbs** and **dump-raw-db** help you list and view the low-level databases within a backend. Unlike the output of other subcommands, the output of the **dump-raw-db** subcommand is neither decrypted nor formatted for readability. Instead, you can see values as they are stored in the backend file.

In a backend database, the `id2entry` index holds LDIF representations of directory entries. For a database that is not encrypted, the corresponding low-level database shows the cleartext strings, as is evident in the following example:

```
$ stop-ds --quiet
$ backendstat list-raw-dbs --backendId dsEvaluation
...
/dc=com,dc=example/id2entry...
...
$ backendstat \
 dump-raw-db \
 --backendId dsEvaluation \
 --dbName /dc=com,dc=example/id2entry
...
Key (len 8):
 00 00 00 00 00 00 00 1E                          ........
 Value (len 437):
 02 00 81 B1 03 01 06 27 75 69 64 3D 62 6A 65 6E  .......'uid=bjen
 73 65 6E 2C 6F 75 3D 50 65 6F 70 6C 65 2C 64 63  sen,ou=People,dc
 3D 65 78 61 6D 70 6C 65 2C 64 63 3D 63 6F 6D 01  =example,dc=com.
 06 11 01 08 01 13 62 6A 65 6E 73 65 6E 40 65 78  ......bjensen@ex
 61 6D 70 6C 65 2E 63 6F 6D 01 09 01 04 30 32 30  ample.com....020
 39 01 16 01 0C 65 6E 2C 20 6B 6F 3B 71 3D 30 2E  9....en, ko;q=0.
 38 01 10 01 29 75 69 64 3D 74 72 69 67 64 65 6E  8...)uid=trigden
 2C 20 6F 75 3D 50 65 6F 70 6C 65 2C 20 64 63 3D  , ou=People, dc=
 65 78 61 6D 70 6C 65 2C 64 63 3D 63 6F 6D 01 04  example,dc=com..
 02 13 50 72 6F 64 75 63 74 20 44 65 76 65 6C 6F  ..Product Develo
 70 6D 65 6E 74 06 50 65 6F 70 6C 65 01 0B 01 07  pment.People....
 42 61 72 62 61 72 61 01 0C 01 0F 2B 31 20 34 30  Barbara....+1 40
 38 20 35 35 35 20 31 38 36 32 01 0D 01 06 4A 65  8 555 1862....Je
 6E 73 65 6E 01 07 02 0E 42 61 72 62 61 72 61 20  nsen....Barbara
 4A 65 6E 73 65 6E 0B 42 61 62 73 20 4A 65 6E 73  Jensen.Babs Jens
 65 6E 01 0E 01 0D 2F 68 6F 6D 65 2F 62 6A 65 6E  en..../home/bjen
 73 65 6E 01 0F 01 0F 2B 31 20 34 30 38 20 35 35  sen....+1 408 55
 35 20 31 39 39 32 01 11 01 04 31 30 30 30 01 12  5 1992....1000..
 01 2E 7B 53 53 48 41 7D 33 45 66 54 62 33 70 37  ..{SSHA}3EfTb3p7
 71 75 6F 75 73 4B 35 34 2B 41 4F 34 71 44 57 6C  quousK54+AO4qDWl
 56 33 4F 39 54 58 48 57 49 4A 49 32 4E 41 3D 3D  V3O9TXHWIJI2NA==
 01 13 01 04 31 30 37 36 01 05 01 14 4F 72 69 67  ....1076....Orig
 69 6E 61 6C 20 64 65 73 63 72 69 70 74 69 6F 6E  inal description
 01 14 01 07 62 6A 65 6E 73 65 6E 01 15 01 0D 53  ....bjensen....S
 61 6E 20 46 72 61 6E 63 69 73 63 6F 01 01 02 01  an Francisco....
 24 38 38 37 37 33 32 65 38 2D 33 64 62 32 2D 33  $887732e8-3db2-3
 31 62 62 2D 62 33 32 39 2D 32 30 63 64 36 66 63  1bb-b329-20cd6fc
 65 63 63 30 35                                   ecc05
...
```

To maintain data confidentiality on disk, you must configure it explicitly. In addition to preventing read access by other users as described in "Setting Up a System Account for a Server", you can configure confidentiality for database backends. When confidentiality is enabled for a backend, the directory server encrypts entries before storing them in the backend.

Important

Encrypting stored directory data does not prevent it from being sent over the network in the clear.

Apply the suggestions in "*Securing Network Connections*" in the *Security Guide* to protect data sent over the network.

Enable backend confidentiality with the default encryption settings as shown in the following example:

```
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set confidentiality-enabled:true \
 --no-prompt \
 --trustAll
```

After confidentiality is enabled, entries are encrypted when next written. That is, the DS directory server does not automatically rewrite all entries in encrypted form. Instead, it encrypts each entry on update, for example, when a user updates their entry or when you import data.

If you import the data again after enabling confidentiality, you can see with the **backendstat dump-raw-db** command that the low-level database for `id2index` no longer contains cleartext:

```
$ stop-ds --quiet
$ [../resources/tests/admin-guide/import_export.bash:#export-backup-ldif]
$ [../resources/tests/admin-guide/import_export.bash:#import-backup-ldif]
$ backendstat \
 dump-raw-db \
 --backendId dsEvaluation \
 --dbName /dc=com,dc=example/id2entry
...
Key (len 8):
 00 00 00 00 00 00 00 C2                          ........
 Value (len 437):
 02 02 81 82 01 C4 95 87 5B A5 2E 47 97 80 23 F4  ........[..G..#.
 CE 5D 93 25 97 D4 13 F9 0A A3 A8 31 9A D9 7A 70  .].%.......1..zp
 FE 3E AC 9D 64 41 EB 7B D5 7F 7E B8 B7 74 52 B8  .>..dA.{.^?~..tR.
 C7 7F C8 79 19 46 7D C5 5D 5B 83 9C 5B 9F 85 28  .^?.y.F}.][..[..(
 83 A2 5F A0 C1 B1 09 FC 2F E3 D8 82 4A AA 8B D9  .._...../...J...
 78 43 34 50 AE A1 52 88 5B 70 97 D2 E1 EA 87 CA  xC4P..R.[p......
```

```
 3B 4D 07 DC F9 F8 30 BB D2 76 51 C8 75 FF FA 80  ;M....0..vQ.u...
 77 E1 6A 8B 5B 8F DA A4 F4 0B B5 20 56 B3 19 19  w.j.[...... V...
 22 D8 9D 38 04 E3 4D 94 A7 99 4B 81 16 AD 88 46  "..8..M...K....F
 FC 3F 7E 78 66 B8 D1 E9 86 A0 F3 AC B6 68 0D A9  .?~xf........h..
 9A A7 3C 30 40 37 97 4E 90 DD 63 16 8E 11 0F 5E  ..<0@7.N..c....^
 9D 5B 86 90 AF 4E E2 1F 9E 70 73 14 0A 11 5C DB  .[...N...ps...\.
 B7 BC B8 A9 31 3F 74 8D 0A 9F F4 6C E1 B0 36 78  ....1?t....l..6x
 F0 5A 5E CD 7C B3 A2 36 66 8E 88 86 A0 8B 9A 77  .Z^.|..6f......w
 D5 CD 7E 9C 4E 62 20 0E D0 DB AD E7 7E 99 46 4F  ..~.Nb .....~.FO
 67 C7 A6 7E 2C 24 82 50 51 9F A7 B2 02 44 5B 30  g..~,$.PQ....D[0
 74 41 99 D9 83 69 EF AE 2E C0 FF C4 E6 4F F2 2F  tA...i.......O./
 95 FB 93 65 30 2A 2D 8D 20 88 83 B5 DE 35 B6 20  ...e0*-. ....5.
 47 17 30 25 60 FD E3 43 B9 D6 A4 F7 47 B6 6C 9F  G.0%`..C....G.l.
 47 FD 63 8E 7F A5 00 CE 6C 3E BC 95 23 69 ED D0  G.c.^?...l>..#i..
 69 4F BE 61 BD 30 C2 40 66 F6 F9 C3 3E C1 D7 8C  iO.a.0.@f...>...
 B0 C8 4A 2E 27 BE 13 6C 40 88 B0 13 A3 12 F4 50  ..J.'..l@......P
 CA 92 D8 EB 4A E5 3F E2 64 A3 76 C7 5C 2B D8 89  ....J.?.d.v.\+..
 A3 6E C1 F7 0A C2 37 7A BD AF 14 4B 52 04 6B F2  .n....7z...KR.k.
 8F 4F C3 F8 00 90 BA 0F EC 6D B1 2D A8 18 0C A6  .O.......m.-....
 29 96 82 3B 5C BC D0 F4 2B BE 9C C5 8B 18 7A DE  )..;\...+.....z.
 C7 B5 10 2D 45 50 4F 77 ED F7 23 34 95 AF C3 2E  ...-EPOw..#4....
 B0 9B FA E9 DF                                   .....
...
```

Similar checks can be run on other low-level databases if you enable confidentiality by backend indexes as described below.

The settings for data confidentiality depend on the encryption capabilities of the JVM. For example, for details about the Sun/Oracle Java implementation, see the explanations in `javax.crypto.Cipher`. You can accept the default settings, or choose to specify the following:

- The cipher algorithm defining how the cleartext is encrypted and decrypted.
- The cipher mode of operation defining how a block cipher algorithm should transform data larger than a single block.
- The cipher padding defining how to pad the cleartext to reach appropriate size for the algorithm.
- The cipher key length, where longer key lengths strengthen encryption at the cost of more performance impact.

The default settings for confidentiality are `cipher-transformation: AES/CBC/PKCS5Padding` and `cipher-key-length: 128`. This means the algorithm is the Advanced Encryption Standard (AES), the cipher mode is Cipher Block Chaining (CBC), and the padding is PKCS#5 padding as described in RFC 2898: PKCS #5: Password-Based Cryptography Specification. The syntax for the `cipher-transformation` is *`algorithm/mode/padding`*, and all three must be specified. When the algorithm does not require a mode, use `NONE`. When the algorithm does not require padding, use `NoPadding`. Use of larger `cipher-key-length` values can require that you install JCE policy files such as those for unlimited strength, as described in "Using Unlimited Strength Cryptography" in the *Security Guide*.

DS servers encrypt data using a symmetric key that is stored with the server configuration. The symmetric key is encrypted in turn with the server's public key that is also stored with the server configuration. When multiple servers are configured to replicate data as described in "Configuring

Replication Settings", the servers replicate the keys as well, allowing any server replica to decrypt the data.

In addition to entry encryption, you can enable confidentiality by backend index, as long as confidentiality is enabled for the backend itself. Confidentiality hashes keys for equality type indexes using SHA-1, and encrypts the list of entries matching a substring key for substring indexes. The following example shows how to enable confidentiality for the `mail` index:

```
$ dsconfig \
 set-backend-index-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name mail \
 --set confidentiality-enabled:true \
 --no-prompt \
 --trustAll
```

After changing the index configuration, you can rebuild the index to enforce confidentiality immediately. For details, see "Configuring and Rebuilding Indexes".

Avoid using sensitive attributes in VLV indexes. Confidentiality cannot be enabled for VLV indexes.

When reading files for an encrypted backend database, be aware that although user data is kept confidential, the following are *not encrypted* on disk:

- As described above, the server encrypts backend database content *the next time it is written*.

  If you configure data confidentiality after initial import, and you must make sure that all relevant data are encrypted, export the data to LDIF and then import the data again. For details, see "*Managing Directory Data*".

- With the exception of the items identified below, backend database file content is encrypted. Yet, the files themselves are not encrypted.

- The `dn2id` index and its keys are not encrypted.

- Substring index keys are not encrypted.

  As described above, the values are encrypted, however.

Encrypting and decrypting data comes with costs in terms of cryptographic processing that reduces throughput and of extra space for larger encrypted values. In general, tests with default settings show that the cost of enabling confidentiality can be quite modest, but your results can vary based on your systems and on the settings used for `cipher-transformation` and `cipher-key-length`. Make sure you test your deployment to qualify the impact of confidentiality before enabling it in production.

Setting Disk Space Thresholds For Database Backends

Directory data growth depends on applications that use the directory. As a result, when directory applications add more data than they delete, the database backend grows until it fills the available disk space. The system can end up in an unrecoverable state if no disk space is available.

Database backends therefore have advanced properties, `disk-low-threshold` and `disk-full-threshold`. When available disk space falls below `disk-low-threshold`, the directory server only allows updates from users and applications that have the `bypass-lockdown` privilege, as described in "About Privileges". When available space falls below `disk-full-threshold`, the directory server stops allowing updates, instead returning an `UNWILLING_TO_PERFORM` error to each update request.

A directory server continues to apply replication updates without regard to the thresholds. The server can therefore fill available disk space despite the thresholds, by accepting replication updates made on other servers. You can give yourself more time to react to the situation both by monitoring directory data growth and also by increasing the thresholds.

If growth across the directory service tends to happen quickly, set the thresholds higher than the defaults to allow more time to react when growth threatens to fill the disk. The following example sets `disk-low-threshold` to 10 GB `disk-full-threshold` to 5 GB for the `dsEvaluation` backend:

```
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set "disk-low-threshold:10 GB" \
 --set "disk-full-threshold:5 GB" \
 --trustAll \
 --no-prompt
```

The properties disk-low-threshold and disk-full-threshold are listed as *advanced* properties. To examine their values with the **dsconfig** command, use the `--advanced` option as shown in the following example:

```
$ dsconfig \
 get-backend-prop \
 --advanced \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --property disk-low-threshold \
 --property disk-full-threshold \
 --trustAll \
 --no-prompt
Property            : Value(s)
--------------------:---------
disk-full-threshold : 5 gb
disk-low-threshold  : 10 gb
```

Automating Entry Expiration and Deletion

If the directory service creates many entries that expire and should be deleted, it is possible to delete the entries by finding them with a time-based search and then deleting them individually. That approach uses replication to delete the entries in all replicas, and works with older versions of the directory service. It has the disadvantage of generating potentially large amounts of replication traffic. With upgraded replicas, you can instead configure the backend database to delete the entries as they expire. This new approach deletes expired entries at the backend database level, without generating replication traffic.

Backend indexes for generalized time (timestamp) attributes have these properties to configure automated, optimized entry expiration and removal:

- ttl-enabled
- ttl-age

Configure this capability by performing the following steps:

1. Prepare an ordering index for a generalized time (timestamp) attribute on entries that expire.

   For details, see "Configuring and Rebuilding Indexes" and "Search: Listing Active Accounts" in the *Developer's Guide*.

2. Using the **dsconfig set-backend-index-prop** command, set `ttl-enabled` on the index to true, and set `ttl-age` on the index to the desired entry lifetime duration.

Once you configure and build the index, the backend can delete expired entries. At intervals of 10 seconds, the backend automatically deletes entries whose timestamps on the attribute are older than the specified lifetime. Entries that expire in the interval between deletions are removed on the next

round. Client applications should therefore check that entries have not expired, as it is possible for expired entries to remain available until the next round of deletions.

When using this capability, keep the following points in mind:

- Entry expiration is per index. The time to live depends on the value of the indexed attribute and the `ttl-age` setting, and all matching entries are subject to TTL.
- If multiple indexes' `ttl-enabled` and `ttl-age` properties are configured, as soon as one of the entry's matching attributes exceeds the TTL, the entry is eligible for deletion.
- The backend deletes the entries directly. Deletion is not visible at the protocol level, and so is not recorded in logs nor returned in persistent searches.

  Furthermore, this means that *deletion is not replicated*. To ensure expired entries are deleted on all replicas, use the same indexes with the same settings on all replicas.
- When a backend deletes an expired entry, the effect is a subtree delete. In other words, if a parent entry expires, the parent entry *and all the parent's child entries* are deleted.

  If you do not want parent entries to expire, index a generalized time attribute that is only present on its child entries.
- The backend deletes expired entries atomically.

  If you update the TTL attribute to prevent deletion and the update succeeds, then TTL has effectively been reset.
- Expiration fails when the `index-entry-limit` is exceeded. (For background information, see "Understanding Index Entry Limits".)

  This only happens if the timestamp for the indexed attribute matches to the nearest millisecond on more than 4000 entries (for default settings). This corresponds to four million timestamp updates per second, which would be very difficult to reproduce in a real directory service.

  It is possible, however, to construct and import an LDIF file where more than 4000 entries have the same timestamp. Make sure not to reuse the same timestamp for thousands of entries when artificially creating entries that you intend to expire.

Updating an Existing Backend to Add a New Base DN

In addition to letting you create new backends as described in "Creating a New Database Backend", DS directory servers let you add a new base DN to an existing backend.

The following example adds the suffix `o=example` to the existing backend `dsEvaluation`:

```
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --add base-dn:o=example \
 --trustAll \
 --no-prompt
$ dsconfig \
 get-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --property base-dn \
 --trustAll \
 --no-prompt
Property : Value(s)
---------:------------------------------
base-dn  : "dc=example,dc=com", o=example
```

Deleting a Database Backend

You delete a database backend by using the **dsconfig delete-backend** command, described in "delete-backend" in the *Configuration Reference*.

When you delete a database backend by using the **dsconfig delete-backend** command, the directory server does not actually remove the database files for two reasons:

1. A mistake could potentially cause lots of data to be lost.
2. Deleting a large database backend could cause severe service degradation due to a sudden increase in I/O load.

Instead, after you run the **dsconfig delete-backend** command you must also manually remove the database backend files.

If you do run the **dsconfig delete-backend** command by mistake and have not yet deleted the actual files, then you can recover from the mistake by creating the backend again, reconfiguring the indexes that were removed, and rebuilding the indexes as described in "Configuring and Rebuilding Indexes".

Chapter 5

Configuring Connection Handlers

This chapter shows you how to configure servers to listen for directory client requests using connection handlers. You can update the configuration using the **dsconfig** command, described in "*dsconfig — manage OpenDJ server configuration*" in the *Reference*.

In this chapter you will learn to:

- Enable client applications to access the directory over LDAP and secure LDAP (LDAPS)
- Enable client applications to access the directory over HTTP whether using DSML, or the REST style
- Enable monitoring using Java Management Extensions (JMX), or over Simple Network Management Protocol (SNMP)
- Enable automated processing of LDIF files
- Configure restrictions for client access such as requiring authentication or limiting the maximum number of concurrent connections
- Configure transport layer security for all relevant protocols

LDAP Client Access

You configure LDAP client access using the **dsconfig** command.

The reserved port number for LDAP is 389. Most examples in the documentation use 1389, which is accessible to non-privileged users.

To Change the LDAP Port Number

1. Change the port number using the **dsconfig** command.

   The following example changes the LDAP port number to 11389:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --set listen-port:11389 \
 --trustAll \
 --no-prompt
```

2. Restart the connection handler so the change takes effect.

   To restart the connection handler, you disable it, then enable it again:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --set enabled:false \
 --trustAll \
 --no-prompt

$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

Preparing For Secure Communications

This section shows how to prepare the server to use a file-based keystore to manage the keys essential to secure communications. For more information about the keys, see "About Certificates, Private Keys, and Secret Keys" in the *Security Guide*.

For instructions on importing trusted certificates on PKCS#11 devices, see the documentation for the device.

Importing Certificates

A client that sets up a secure connection with a server must be able to trust the server certificate. A server that uses mutual authentication (checking the client certificate) must be able to trust the client certificate. In either case, this involves finding the signing certificate in a keystore or a truststore.

In some cases, DS servers act as clients of external services. For example, REST to LDAP can resolve OAuth 2.0 tokens by sending secure requests to an authorization server. The server can also connect to another LDAP server when using pass-through authentication.

The default Java truststore contains signing certificates from well-known CAs. If the CA certificate is not in the default truststore, or the certificate is self-signed, then you can import it into a truststore as described here.

This section includes the following procedures:

- "To Import a Trusted Client Certificate"
- "To Import the Server Certificate"
- "To Import a Trusted CA Certificate"

To Import a Trusted Client Certificate

The following steps demonstrate using the **keytool** command to add a client application's binary format, self-signed certificate to a new truststore for the DS server. This procedure enables the DS server to recognize a self-signed client application certificate when negotiating a secure connection. To allow a client application to perform an LDAP bind using its certificate, see "Authenticating Client Applications With a Certificate" in the *Developer's Guide* instead.

1. Import the self-signed client certificate:

```
$ keytool \
 -import \
 -trustcacerts \
 -alias myapp-cert \
 -file myapp-cert.pem \
 -keystore /path/to/opendj/config/truststore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12 \
 -noprompt
Certificate was added to keystore
```

   In this example, the new truststore uses the same PIN as the default keystore.

2. Add a trust manager provider to access the truststore:

```
$ dsconfig \
 create-trust-manager-provider \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --type file-based \
 --provider-name "PKCS12 Trust Manager" \
 --set enabled:true \
 --set trust-store-file:/path/to/opendj/config/truststore \
 --set trust-store-pin:"&{file:/path/to/opendj/config/keystore.pin}" \
 --trustAll \
 --no-prompt
```

3. Configure connection handlers to set the trust manager provider as shown, for example in "LDAP Client Access With Transport Layer Security" and in "To Set Up HTTPS Access".

To Import the Server Certificate

If your client uses a Java truststore, import the DS server signing certificate using the **keytool** command.

1. Export the server certificate from the server keystore.

   The following example shows the **keytool** command to export the self-signed server certificate in binary format. Notice that the keystore PIN is stored in a file, which is the default setting:

```
$ keytool \
 -export \
 -rfc \
 -alias server-cert \
 -file server-cert.pem \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12
Certificate stored in file <server-cert.pem>
```

2. Import the server certificate into the client truststore:

```
$ keytool \
 -import \
 -trustcacerts \
 -alias server-cert \
 -file server-cert.pem \
 -keystore my-keystore \
 -storetype PKCS12 \
 -storepass changeit \
 -noprompt
Certificate was added to keystore
```

To Import a Trusted CA Certificate

If you use self-signed certificates, or if your CA is not well-known, you can nevertheless import the certificate as a CA certificate into a Java file-based truststore using the **keytool** command. The application that relies on the truststore can then trust the certificate in the same way it trusts well-known CA certificates.

Follow these steps to import the CA certificate into a truststore for the server:

1. Import the certificate as a CA certificate:

```
$ keytool \
 -import \
 -trustcacerts \
 -alias ca-cert \
 -file ca.crt \
 -keystore /path/to/opendj/config/truststore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12 \
 -noprompt
Certificate was added to keystore
```

2. Make sure the server can use the truststore in one of the following ways:

   - If no trust manager provider is configured to access the truststore, add one as shown in "To Import a Trusted Client Certificate".
   - If a trust manager provider is already configured to access the truststore, restart the server to force it to reload the truststore:

     ```
     $ stop-ds --restart --quiet
     ```

     If necessary, you can avoid restarting the server by disabling the trust manager provider and then enabling it again.

3. Configure connection handlers to set the trust manager provider as shown, for example in "LDAP Client Access With Transport Layer Security" and in "To Set Up HTTPS Access".

Setting Up Server Certificates

When you install the DS server, you can choose to configure secure connections and either generate a key pair with a self-signed certificate, or import your own keystore. The default PKCS#12 keystore is `/path/to/opendj/config/keystore`, and the self-signed public key certificate has the alias `server-cert`. The password for the keystore and the private key is stored in cleartext in the file `/path/to/opendj/config/keystore.pin`.

If you chose to set up a secure connection as part of the installation process, you can skip this section.

This section includes the following procedures:

- "To Set Up a CA-Signed Certificate"
- "To Set Up a Self-Signed Certificate"
- "To Use an Alternative Keystore Implementation"

To Set Up a CA-Signed Certificate

This procedure shows how to prepare a new key pair with a CA-signed certificate for use in setting up secure connections.

The high-level steps to perform are the following:

- Generate a server private key and public key certificate in your keystore.
- Issue a signing request to the CA, who responds with a CA-signed certificate.
- Import the CA-signed certificate where appropriate.
- Set up a key manager provider to use the keystore.

A detailed example follows:

1. Prepare the password for the keystore.

   By default, DS servers are configured to hold the password in a file, `/path/to/opendj/config/keystore.pin`:

   ```
   $ touch /path/to/opendj/config/keystore.pin
   $ chmod 600 /path/to/opendj/config/keystore.pin
   # Add keystore and private key password in cleartext on a single line:
   $ vi /path/to/opendj/config/keystore.pin
   ```

2. Generate the server key pair (private key and public key certificate) using the Java **keytool** command.

   One step in verifying the certificate's validity is checking that the subject's FQDN matches the FQDN obtained from DNS.

   The FQDN for the DS server, visible in the **status** command output, is set both as a `DNSName` in the certificate's `SubjectAlternativeName` list, and also in the CN of the certificate's subject name DN for backwards compatibility:

```
$ keytool \
 -genkeypair \
 -alias server-cert \
 -ext "san=dns:opendj.example.com" \
 -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -keypass:file /path/to/opendj/config/keystore.pin
```

Note

Notice that the `-storepass` and `-keypass` options take identical password arguments. DS servers use the same password to protect the keystore and the private key.

If the server can respond on multiple FQDNs, then specify multiple subject alternative names when using the **keytool** command's `-ext` option. In the following example the primary FQDN is `opendj.example.com` and the alternative is `ldap.example.com`:

```
$ keytool \
 -genkeypair \
 -alias server-cert \
 -ext "san=dns:opendj.example.com,dns:ldap.example.com" \
 -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -keypass:file /path/to/opendj/config/keystore.pin
```

For an example showing how to use a wildcard certificate, see "To Set Up a Key Pair With a Wildcard Certificate".

3. Create a certificate signing request file for the generated certificate:

```
$ keytool \
 -certreq \
 -alias server-cert \
 -file server-cert.csr \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin
```

4. Have the CA sign the request (`server-cert.csr`).

   See the instructions from your CA on how to provide the request.

   The CA returns the signed certificate.

5. (Optional) If you have set up your own CA and signed the certificate, or are using a CA whose signing certificate is not included in the Java runtime environment, import the CA certificate into the keystore so that it can be trusted.

   Otherwise, when you import the signed certificate from an (unknown) CA, the **keytool** command fails to import the signed certificate with the message `keytool error: java.lang.Exception: Failed to establish chain from reply`.

   For an example command, see "To Import a Trusted CA Certificate".

6. Import the signed certificate from the CA reply into the keystore where you generated the server certificate.

   In this example the certificate from the reply is `server-cert.crt`:

   ```
   $ keytool \
    -import \
    -alias server-cert \
    -file server-cert.crt \
    -keystore /path/to/opendj/config/keystore \
    -storetype PKCS12 \
    -storepass:file /path/to/opendj/config/keystore.pin \
    -keypass:file /path/to/opendj/config/keystore.pin \
    -noprompt
   ```

7. Configure the file-based key manager provider for the keystore that you set up with the **keytool** command:

   ```
   $ dsconfig \
    set-key-manager-provider-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --provider-name "Default Key Manager" \
    --set enabled:true \
    --trustAll \
    --no-prompt
   ```

   If you stored the keystore password somewhere besides the file, `/path/to/opendj/config/keystore.pin`, shown in the examples in this procedure, also adjust the `key-store-*` settings accordingly.

   At this point, the DS server can use the CA-signed certificate.

8. If you use a CA certificate that is not known to clients, such as a CA that you set up yourself rather than a well-known CA, import the CA certificate into the client application truststore. For an example command, see "To Import a Trusted CA Certificate".

   Otherwise the client application cannot trust the signature on the server certificate.

To Set Up a Self-Signed Certificate

This procedure shows how to prepare a new key pair with a self-signed certificate for use in setting up secure connections.

The high-level steps to perform are the following:

- Generate a server private key and public key certificate in your keystore.
- Self-sign the certificate.
- Set up a key manager provider to use the keystore.

To *replace the existing server key pair with a self-signed certificate and new private key*, first, use **keytool -delete -alias server-cert** to delete the existing keys, then generate a new key pair with the same alias. Either reuse the existing password in `keystore.pin`, or use a new password as shown in the steps below.

1. Prepare the password for the keystore.

   By default, DS servers are configured to hold the password in a file, `/path/to/opendj/config/keystore.pin`:

   ```
   $ touch /path/to/opendj/config/keystore.pin
   $ chmod 600 /path/to/opendj/config/keystore.pin
   # Add keystore and private key password in cleartext on a single line:
   $ vi /path/to/opendj/config/keystore.pin
   ```

2. Generate the key pair using the Java **keytool** command:

   ```
   $ keytool \
    -genkeypair \
    -alias server-cert \
    -ext "san=dns:opendj.example.com" \
    -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/opendj/config/keystore \
    -storetype PKCS12 \
    -storepass:file /path/to/opendj/config/keystore.pin \
    -keypass:file /path/to/opendj/config/keystore.pin
   ```

   In this example, the DS server runs on a system with FQDN `opendj.example.com`. The keystore is created in the server `config` directory.

Note

Notice that the `-storepass` and `-keypass` options take identical password arguments. DS servers use the same password to protect the keystore and the private key.

If the server can respond on multiple FQDNs, then specify multiple subject alternative names when using the **keytool** command's `-ext` option. In the following example the primary FQDN is `opendj.example.com` and the alternative is `ldap.example.com`:

```
$ keytool \
 -genkeypair \
 -alias server-cert \
 -ext "san=dns:opendj.example.com,dns:ldap.example.com" \
 -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -keypass:file /path/to/opendj/config/keystore.pin
```

For an example showing how to use a wildcard certificate, see "To Set Up a Key Pair With a Wildcard Certificate".

3. Self-sign the server certificate:

```
$ keytool \
 -selfcert \
 -alias server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin
```

4. Configure the file-based key manager provider to access the keystore with keystore/private key password.

   In this example, the alias is `server-cert` and the password is in the file, `/path/to/opendj/config/keystore.pin`.

   If you are replacing a key pair with a self-signed certificate, reusing the `server-cert` alias and password stored in `keystore.pin`, then you can skip this step:

```
$ dsconfig \
 set-key-manager-provider-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Default Key Manager" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

If you stored the keystore password somewhere besides the file, `/path/to/opendj/config/keystore.pin`, shown in the examples in this procedure, also adjust the `key-store-*` settings accordingly.

At this point, the DS server can use the self-signed certificate, for example, for StartTLS and LDAPS or HTTPS connection handlers.

To Use an Alternative Keystore Implementation

To use an alternative keystore implementation, start with a different keystore type when generating the keypair:

1. Use one of the keystore types supported by the Java runtime environment:

   Java Keystore

   The basic Java keystore type is `JKS`:

   ```
   $ keytool \
    -genkeypair \
    -alias server-cert \
    -ext "san=dns:opendj.example.com" \
    -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/opendj/config/keystore.jks \
    -storetype JKS \
    -storepass:file /path/to/opendj/config/keystore.pin \
    -keypass:file /path/to/opendj/config/keystore.pin
   ```

   This is the keystore type if you do not specify a `-storetype` option.

   Java Cryptography Extension Keystore

   The `JCEKS` type lets you take advantage of additional Java cryptography extensions and stronger protection for private keys:

```
$ keytool \
-genkeypair \
-alias server-cert \
-ext "san=dns:opendj.example.com" \
-dname "CN=opendj.example.com,O=Example Corp,C=FR" \
-keystore /path/to/opendj/config/keystore.jceks \
-storetype JCEKS \
-storepass:file /path/to/opendj/config/keystore.pin \
-keypass:file /path/to/opendj/config/keystore.pin
```

LDAP Keystore

DS servers implement an `OpenDJ` security provider for LDAP and LDIF-based keystore services. Its keystore type is `LDAP`.

For details, see "Using an LDAP Keystore" in the *Security Guide*.

PKCS#11 device

A PKCS#11 device, such as an HSM, can be used as a keystore.

For details, see "Using a Hardware Security Module" in the *Security Guide*.

PKCS#12 Keystore

The `PKCS12` type lets you use a PKCS#12 format file. This is the default for DS servers. It is a standard format and is interoperable with other systems that do not necessarily depend on a Java runtime environment:

```
$ keytool \
-genkeypair \
-alias server-cert \
-ext "san=dns:opendj.example.com" \
-dname "CN=opendj.example.com,O=Example Corp,C=FR" \
-keystore /path/to/opendj/config/keystore \
-storetype PKCS12 \
-storepass:file /path/to/opendj/config/keystore.pin \
-keypass:file /path/to/opendj/config/keystore.pin
```

2. After using an alternate keystore type, make sure that you set up:

   - The key manager provider to open the correct keystore with the correct credentials.

     Any components using that key manager provider to use the correct certificate alias.

Working With Test Key Pairs

This section includes procedures for working with test key pairs. The procedures use the **openssl** command to perform actions that are not necessarily possible with the Java **keytool** command. OpenSSL software is available from https://www.openssl.org/.

Important

OpenSSL software is independent from and not associated with the DS project. The examples shown in this section might not work with your version of OpenSSL. See the **openssl** documentation for your version.

Examples in this section were originally written with OpenSSL version 1.0.2.

This section includes the following procedures:

- "To Set Up a CA Signing Certificate"
- "To Sign a Certificate Signing Request"
- "To Set Up a Key Pair With a Wildcard Certificate"

To Set Up a CA Signing Certificate

Follow these steps to set up a CA key pair with a signing certificate:

1. Generate a private key for the CA:

```
$ openssl \
 genpkey \
 -algorithm RSA \
 -out ca.key \
 -pkeyopt rsa_keygen_bits:4096
```

2. Self-sign a corresponding certificate:

```
$ openssl \
 req \
 -new \
 -x509 \
 -days 7300 \
 -subj "/C=FR/O=Example Corp/CN=example.com" \
 -key ca.key \
 -out ca.crt
```

The signing certificate is `ca.crt`, and it corresponds to the private key `ca.key`.

To Sign a Certificate Signing Request

Follow these steps to use a CA signing certificate to sign a certificate signing request (CSR):

1. If you have not already done so, set up a CA key pair as described in "To Set Up a CA Signing Certificate".
2. Obtain a CSR file for the certificate to sign.
3. Create a signed certificate from the CSR.

   The following example preserves the SAN for a server certificate generated using the **keytool** command and the `-ext "san=dns:opendj.example.com"` option:

   ```
   $ openssl \
    x509 \
    -req \
    -in server-cert.csr \
    -CA ca.crt \
    -CAkey ca.key \
    -CAcreateserial \
    -extfile \
     <(cat /etc/ssl/openssl.cnf \
     <(printf "[SAN]\nsubjectAltName=DNS:opendj.example.com")) \
    -extensions SAN \
    -out server-cert.crt
   ```

   The following example preserves the SAN for a wildcard certificate as described in "To Set Up a Key Pair With a Wildcard Certificate":

   ```
   $ openssl \
    x509 \
    -req \
    -in example.com.csr \
    -CA ca.crt \
    -CAkey ca.key \
    -CAcreateserial \
    -extfile \
     <(cat /etc/ssl/openssl.cnf \
     <(printf "[SAN]\nsubjectAltName=DNS:*.example.com")) \
    -extensions SAN \
    -out example.com.crt
   ```

   In both examples, the certificates to return to the requestors are the `.crt` files.

To Set Up a Key Pair With a Wildcard Certificate

A wildcard certificate uses a `*` to replace the top-level subdomain in the subject FQDN, and can list domains in the subject alternative domain list.

The FQDN for a server, visible in the **status** command output, must also match a `DNSName` value or pattern in the certificate's `SubjectAlternativeName` list.

Follow these steps to set up a key pair with a wildcard certificate:

1. Generate a private key:

```
$ openssl \
 genpkey \
 -algorithm RSA \
 -pkeyopt rsa_keygen_bits:2048 \
 -out example.com.key
```

2. Generate a certificate signing request for the CA:

```
$ openssl \
 req \
 -new \
 -sha256 \
 -key example.com.key \
 -subj "/C=FR/O=Example Corp/CN=*.example.com" \
 -reqexts SAN \
 -config \
  <(cat /etc/ssl/openssl.cnf \
  <(printf "[SAN]\nsubjectAltName=DNS:*.example.com")) \
 -out example.com.csr
```

3. Get the CA to sign the request and return the certificate.

   For an example of how to do this yourself, see "To Sign a Certificate Signing Request".

4. (Optional) Convert the results into a PKCS#12 format Java keystore file.

```
$ openssl \
 pkcs12 \
 -export \
 -in example.com.crt \
 -inkey example.com.key \
 -name server-cert \
 -CAfile ca.crt \
 -password file:/path/to/opendj/config/keystore.pin \
 -out /path/to/opendj/config/keystore
```

   This keystore can be used for testing the wildcard certificate with the DS server as in the following example:

```
$ ldapsearch \
 --port 1636 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useSSL \
 "(uid=bjensen)" \
 cn
The server is using the following certificate:
    Subject DN:  CN=*.example.com, O=Example Corp, C=FR
    Issuer DN:  CN=example.com, O=Example Corp, C=FR
    Validity:  <validity-dates>
Do you wish to trust this certificate and continue connecting to the server?
Please enter "yes" or "no":yes
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
```

LDAP Client Access With Transport Layer Security

StartTLS negotiations start on the unsecure LDAP port, and then protect communication with the client. You can configure StartTLS when setting up a server, or later using the **dsconfig** command.

To Enable StartTLS on the LDAP Port

1. Make sure you have a server certificate installed:

```
$ keytool \
 -list \
 -alias server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12
...
server-cert, <date>, PrivateKeyEntry
...
```

2. Activate StartTLS on the current LDAP port:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --set allow-start-tls:true \
 --set key-manager-provider:"Default Key Manager" \
 --set trust-manager-provider:"JVM Trust Manager" \
 --trustAll \
 --no-prompt
```

The change takes effect. No need to restart the server.

LDAP Client Access Over SSL

An LDAP connection handler configured to use LDAPS (LDAP/SSL) allows only secure connections from client applications. You can configure LDAPS when setting up a server, or later using the **dsconfig** command.

The reserved port number for LDAPS is 636. Most examples in the documentation use 1636, which is accessible to non-privileged users.

To Set Up LDAPS Access

If LDAPS access was not configured at during setup, follow these steps:

1. Make sure a server certificate and associated private key are available:

```
$ keytool \
 -list \
 -alias server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12
server-cert, <date>, PrivateKeyEntry
```

2. Configure the server to activate LDAPS access:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --set enabled:true \
 --set listen-port:1636 \
 --set use-ssl:true \
 --trustAll \
 --no-prompt
```

3. (Optional) If the deployment requires SSL client authentication, set the properties `ssl-client-auth-policy` and `trust-manager-provider` appropriately.

To Change the LDAPS Port Number

1. Change the port number using the **dsconfig** command.

   The following example changes the LDAPS port number to 11636:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --set listen-port:11636 \
 --trustAll \
 --no-prompt
```

2. Restart the connection handler so the change takes effect.

   To restart the connection handler, you disable it, then enable it again:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --set enabled:false \
 --trustAll \
 --no-prompt

$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

Restricting Client Access

Using a server's global configuration properties, you can add global restrictions on how clients access the server. These settings are per server, and so must be set independently on each server.

These global settings are fairly coarse-grained. For a full discussion of the rich set of administrative privileges and fine-grained access control instructions that DS servers support, see "*Configuring Privileges and Access Control*".

Consider the following global configuration settings:

bind-with-dn-requires-password

Whether the server should reject any simple bind request that contains a DN but no password. Default: `true`

To change this setting use the following command:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set bind-with-dn-requires-password:false \
 --no-prompt \
 --trustAll
```

max-allowed-client-connections

Restricts the number of concurrent client connections to this server. Default: 0, meaning no limit is set.

To set a limit of 64K use the following command:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set max-allowed-client-connections:65536 \
 --no-prompt \
 --trustAll
```

allowed-client
denied-client

Restrict which clients can connect to the server.

restricted-client
restricted-client-connection-limit

Restrict the number of concurrent connections per client.

unauthenticated-requests-policy

This setting can take the following values:

`reject`

Reject requests (other than bind or StartTLS requests) received from a client that has not yet been authenticated, whose last authentication attempt was unsuccessful, or whose last authentication attempt used anonymous authentication.

`allow-discovery`

Like `reject`, but allows unauthenticated base object searches of the root DSE.

This setting supports applications that read the root DSE to discover server capabilities, and applications that target the root DSE for keep-alive heartbeats.

`allow`

Allow all unauthenticated requests, subject to privileges and access control.

To prevent anonymous binds except to read the root DSE, use the following command:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set unauthenticated-requests-policy:allow-discovery \
 --no-prompt \
 --trustAll
```

return-bind-error-messages

Does not restrict access, but by default prevents a server from returning extra information about why a bind failed, as that information could be used by an attacker. Instead, the information is written to the server errors log. Default: `false`.

To have the server return additional information about why a bind failed use the following command:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set return-bind-error-messages:true \
 --no-prompt \
 --trustAll
```

TLS Protocols and Cipher Suites

When a server and client negotiate a secure connection, they negotiate use of a common protocol and cipher suite. If they cannot negotiate a common protocol and cipher suite, they will fail to set up a secure connection.

Furthermore, researchers continue to find vulnerabilities in protocols and cipher suites. If the server continues to support those protocols and cipher suites, then clients can use them when negotiating connections. Attackers can then exploit the vulnerabilities. It is therefore important to understand how to read and to restrict the list of supported protocols and cipher suites.

DS servers depend on the underlying JVM to support security protocols and cipher suites. By default, all supported protocols and cipher suites are available. For details, see the documentation for the JVM. For Oracle Java, see the *Java Cryptography Architecture Oracle Providers Documentation* describing the The `SunJSSE` Provider.

Bear in mind that support for protocols and cipher suites can be added and removed in Java update releases. The protocols and cipher suites available are also determined by the security policy. For example, see "Using Unlimited Strength Cryptography" in the *Security Guide*.

This section includes the following procedures:

- "To List Protocols and Cipher Suites"
- "To Restrict Protocols and Cipher Suites"
- "To Restrict Protocols For Command-Line Tools"

To List Protocols and Cipher Suites

- To list the protocols and cipher suites that DS servers support, read the `supportedTLSProtocols` and `supportedTLSCiphers` attributes of the root DSE:

```
$ ldapsearch \
 --port 1389 \
 --baseDN "" \
 --searchScope base \
 "(objectclass=*)" \
 supportedTLSCiphers supportedTLSProtocols
```

  A `supportedTLSCiphers` name, such as `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`, identifies the key attributes of the cipher suite:

  `TLS`

  Specifies the protocol, in this case TLS.

  `ECDHE_RSA`

  Specifies the key exchange algorithm used to determine how the client and server authenticate during the handshake phase.

  In this example, an elliptic curve variant of the Diffie-Hellman key exchange is used, where a random value chosen by the client is encrypted with the server's RSA public key.

  `WITH_AES_256_GCM`

  Specifies the bulk encryption algorithm, including the key size or initialization vectors.

  This example specifies the Advanced Encryption Standard (AES) with 256-bit key size and Galois/Counter Mode (GCM) block cipher mode.

  `SHA384`

  Specifies the message authentication code algorithm used to create the message digest, which is a cryptographic hash of each block in the message stream.

  In this example, the SHA-2 hash function, SHA-384, is used.

  A `supportedTLSProtocols` name identifies the protocol and version, such as `TLSv1.2`.

To Restrict Protocols and Cipher Suites

Choosing which protocol versions and cipher suites to allow for negotiating secure connections depends on known vulnerabilities, estimations of what is secure, and what peers support.

The default support is backward-compatible with old clients, meaning that if you do nothing it is possible for a client to use a protocol version that is known to have vulnerabilities, or to negotiate an insecure or very weakly secure connection. To avoid this, limit the protocols and cipher suites that you allow.

You can limit supported protocols and cipher suites by setting the properties, `ssl-protocol` and `ssl-cipher-suite`. These are available on connection handlers, and on components that connect to remote services including the global Crypto Manager component used for replication.

This procedure is based on server-side TLS recommendations from the Mozilla Operations Security team taken at the time of this writing. Recommendations evolve. Make sure you use current recommendations when configuring security settings.

The examples in this procedure assume you have installed an unlimited encryption strength policy:

1. For each cipher suite key algorithm to support, create a key pair using the supported key algorithm.

   The following example adds two key pairs to the default PKCS#12 keystore, with one key using RSA and the other key using elliptic curve.

   ```
   $ keytool \
    -genkeypair \
    -alias server-cert-rsa \
    -keyalg RSA \
    -ext "san=dns:opendj.example.com" \
    -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/opendj/config/keystore \
    -storetype PKCS12 \
    -storepass:file /path/to/opendj/config/keystore.pin \
    -keypass:file /path/to/opendj/config/keystore.pin
   $ keytool \
    -genkeypair \
    -alias server-cert-ec \
    -keyalg EC \
    -ext "san=dns:opendj.example.com" \
    -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/opendj/config/keystore \
    -storetype PKCS12 \
    -storepass:file /path/to/opendj/config/keystore.pin \
    -keypass:file /path/to/opendj/config/keystore.pin
   ```

2. On the components you use, explicitly set the supported protocols and cipher suites.

   The following example adjusts settings for the LDAP and LDAPS connection handlers:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --add ssl-protocol:TLSv1.2 \
 --add ssl-cipher-suite:TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 \
 --add ssl-cipher-suite:TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 \
 --add ssl-cipher-suite:TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 \
 --add ssl-cipher-suite:TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 \
 --add ssl-cipher-suite:TLS_EMPTY_RENEGOTIATION_INFO_SCSV \
 --no-prompt \
 --trustAll
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --add ssl-protocol:TLSv1.2 \
 --add ssl-cipher-suite:TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 \
 --add ssl-cipher-suite:TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 \
 --add ssl-cipher-suite:TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 \
 --add ssl-cipher-suite:TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 \
 --add ssl-cipher-suite:TLS_EMPTY_RENEGOTIATION_INFO_SCSV \
 --no-prompt \
 --trustAll
```

The `TLS_ECDHE_EC*` cipher suites call for a server certificate with an elliptic curve key algorithm. The `TLS_ECDHE_RSA*` cipher suites call for a server certificate with an RSA key algorithm.

The `TLS_EMPTY_RENEGOTIATION_INFO_SCSV` cipher suite is a renegotiation information extension with a special Signaling Cipher Suite Value (SCSV) to help older clients properly complete a handshake.

3. On the components you use, set the multivalued property, `ssl-cert-nickname`, to identify each certificate alias:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --set enabled:true \
 --set listen-port:1389 \
 --set allow-start-tls:true \
 --set ssl-cert-nickname:server-cert-ec \
 --set ssl-cert-nickname:server-cert-rsa \
 --set key-manager-provider:"Default Key Manager" \
 --set trust-manager-provider:"JVM Trust Manager" \
```

```
 --trustAll \
 --no-prompt
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --set listen-port:1636 \
 --set enabled:true \
 --set use-ssl:true \
 --set ssl-cert-nickname:server-cert-ec \
 --set ssl-cert-nickname:server-cert-rsa \
 --trustAll \
 --no-prompt
```

When negotiating a secure connection, the server can now use either key.

To Restrict Protocols For Command-Line Tools

You can specify which protocol versions to allow when command-line tools negotiate secure connections with LDAP servers.

The command-line tools depend on a system property, `org.opends.ldaps.protocols`. This property takes a comma-separated list of protocols. The default is constructed from the list of all protocols the JVM supports, removing protocol names starting with `SSL`. For example, if support is enabled in the JVM for versions 1.0, 1.1, and 1.2 of the TLS protocol, then the default is `"TLSv1,TLSv1.1,TLSv1.2"`.

- Restrict the protocols to use by setting the property in one of the following ways:
  - Set the property by editing the `java-args` for the command in `config/java.properties`.

    For example, to restrict the protocol to TLS v1.2 when the **status** command negotiates a secure administrative connection, edit the corresponding line in `config/java.properties`:

    ```
    status.java-args=-Xms8m -client -Dorg.opends.ldaps.protocols=TLSv1.2
    ```

  - Set the property at runtime when running the command.

    The following example restricts the protocol to TLS v1.2 when the **status** command negotiates a secure administrative connection:

    ```
    $ export OPENDJ_JAVA_ARGS="-Dorg.opends.ldaps.protocols=TLSv1.2"
    $ status \
     --bindDn "cn=Directory Manager" \
     --bindPassword password \
     --hostname opendj.example.com \
     --port 4444 \
     --usePkcs12TrustStore /path/to/opendj/config/keystore \
     --trustStorePasswordFile /path/to/opendj/config/keystore.pin
    ```

Client Certificate Validation and the Directory

This section clarifies the roles that client applications' X.509 digital certificates play in establishing secure connections and in authenticating the client as a directory user. Be aware that establishing a secure connection happens before the server handles the LDAP or HTTP requests that the client sends over the secure connection. Establishing a secure connection is handled separately from authenticating a client as a directory user, even though both processes can involve the client's certificate.

When a client and a server negotiate a secure connection over LDAPS or HTTPS, or over LDAP using the StartTLS operation, they can use public key cryptography to authenticate each other. The server, client, or both present certificates to each other. By default, DS LDAPS and HTTPS connection handlers are configured to present the server certificate, and to consider the client certificate optional. The connection handler property `ssl-client-auth-policy` makes the latter behavior configurable. For the DSML and REST to LDAP gateways, HTTPS negotiation is handled by the web application container where the gateway runs. See the web application container documentation for details on configuring how the container handles the client certificate.

One step toward establishing a secure connection involves validating the certificate that was presented by the other party. Part of this is trusting the certificate. The certificate identifies the client or server and the CA certificate used to sign the client or server certificate. The validating party checks that the other party corresponds to the one identified by the certificate, and checks that the signature can be trusted. If the signature is valid, and the CA certificate used to sign the certificate can be trusted, then the certificate can be trusted. This part of the validation process is also described briefly in "How Keys are Used" in the *Security Guide*.

Certificates can be revoked after they are signed. Therefore, the validation process can involve checking whether the certificate is still valid. Two different methods for performing this validation use the Online Certificate Status Protocol (OCSP) or Certificate Revocation Lists (CRLs). OCSP is a newer solution that provides an online service to handle the revocation check for a specific certificate. CRLs are potentially large lists of user certificates that are no longer valid or that are on hold. A CRL is signed by the CA. The validating party obtains the CRL and checks that the certificate being validated is not listed. For a brief comparison, see *OCSP: Comparison to CRLs*. A certificate can include links to contact the OCSP responder or to the CRL distribution point. The validating party can use these links to check whether the the certificate is still valid.

In both cases, the CA who signed the certificate acts as the OCSP responder or publishes the CRLs. When establishing a secure connection with a client application, the server relies on the CA for OCSP and CRLs. This is the case even when the DS server is the repository for the CRLs.

DS directory services are logical repositories for certificates and CRLs. For example, your DS directory service can store CRLs in a `certificateRevocationList` attribute as in the following example entry:

```
dn: cn=My CA,dc=example,dc=com
objectClass: top
objectClass: applicationProcess
objectClass: certificationAuthority
cn: My CA
authorityRevocationList;binary: Base64-encoded ARL
cACertificate;binary:: Base64-encoded CA certificate
certificateRevocationList;binary:: Base64-encoded CRL
```

The CRL could then be replicated to other DS directory servers for high availability. (Notice the ARL in this entry. An ARL is like a CRL, but for CA certificates.)

Again, despite being a repository for CRLs, the DS directory service does not use the CRLs directly when checking a client certificate. Instead, when negotiating a secure connection, the server depends on the JVM security configuration. The JVM configuration governs whether validation uses OCSP, CRLs, or both. As described in the *Java PKI Programmer's Guide* under *Support for the CRL Distribution Points Extension*, and *Appendix C: On-Line Certificate Status Protocol (OCSP) Support*, the JVM relies on system properties that define whether to use the CRL distribution points defined in certificates, and how to handle OCSP requests. These system properties can be set system-wide in `$JAVA_HOME/lib/security/java.security` (`$JAVA_HOME/jre/lib/security/java.security` for JDK 8). The JVM handles revocation checking without the DS server's involvement.

After a connection is negotiated, the server can authenticate a client application at the LDAP level based on the certificate. For details, see "Authenticating Client Applications With a Certificate" in the *Security Guide*.

OCSP and obtaining CRLs depend on network access to the CA. If DS servers or the DSML or REST to LDAP gateways run on a network where the CA is not accessible, and the deployment nevertheless requires OSCP or checking CRLs for client application certificates, then you must provide some alternative means to handle OCSP or CRL requests. The JVM can be configured to use a locally available OCSP responder, for example, and that OCSP responder might depend on DS servers. If the solution depends on CRLs, you could regularly update the CRLs in the directory with copies of the CA CRLs obtained by other means.

RESTful Client Access Over HTTP

DS software offers two ways to give RESTful client applications HTTP access to directory user data as JSON resources:

- Enable the listener on a server to respond to REST requests.

  With this approach, you do not need to install additional software.

  For details, see the following procedures:

  - "To Set Up HTTP Access"

- "To Set Up HTTPS Access"
- "To Set Up REST Access to User Data"
- "To Set Up HTTP Authorization"

- Configure the external REST to LDAP gateway Servlet to access the directory service.

  With this approach, you must install the gateway separately.

  For details, see "To Set Up REST to LDAP Gateway".

DS servers also expose administrative data over HTTP. For details, see "To Set Up REST Access to Administrative Data".

The REST to LDAP mappings follow these rules to determine JSON property types:

- If the LDAP attribute is defined in the LDAP schema, then the REST to LDAP mapping uses the most appropriate type in JSON. For example, numbers appear as JSON numbers, and booleans as booleans.
- If the LDAP attribute only has one value, then it is returned as a scalar.
- If the LDAP attribute has multiple values, then the values are returned in an array.

To Set Up HTTP Access

DS servers have a handler for HTTP connections. This handler exposes directory data over HTTP, including the RESTful API demonstrated in "*Performing RESTful Operations*" in the *Developer's Guide*. The HTTP connection handler can be enabled during the setup process.

This procedure shows you how to enable the HTTP connection handler if you did not enable the connection handler during the setup process.

The HTTP connection handler exposes directory data at HTTP endpoints. After you set up the HTTP connection handler, make sure that at least one HTTP endpoint is enabled, for example by following the steps described in "To Set Up REST Access to User Data", or the steps described in "To Set Up REST Access to Administrative Data". It is possible to enable multiple HTTP endpoints, as long as their base paths are different.

1. If you did not enable the HTTP connection handler during the setup process with the `--httpPort` or `--httpsPort` option, create the connection handler configuration:

```
$ dsconfig \
 create-connection-handler \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name HTTP \
 --type http \
 --set enabled:true \
 --set listen-port:8080 \
 --no-prompt \
 --trustAll
```

2. (Optional) Enable an HTTP access log.

   - The following command enables JSON-based HTTP access logging as described in "Configuring JSON Access Logs":

     ```
     $ dsconfig \
      set-log-publisher-prop \
      --hostname opendj.example.com \
      --port 4444 \
      --bindDN "cn=Directory Manager" \
      --bindPassword password \
      --publisher-name "Json File-Based HTTP Access Logger" \
      --set enabled:true \
      --no-prompt \
      --trustAll
     ```

   - The following command enables HTTP access logging as described in "Standard HTTP Access Logs":

     ```
     $ dsconfig \
      set-log-publisher-prop \
      --hostname opendj.example.com \
      --port 4444 \
      --bindDN "cn=Directory Manager" \
      --bindPassword password \
      --publisher-name "File-Based HTTP Access Logger" \
      --set enabled:true \
      --no-prompt \
      --trustAll
     ```

3. (Optional) If necessary, change the connection handler configuration using the **dsconfig** command.

To Set Up HTTPS Access

The following steps demonstrate how to change the default HTTP connection handler configuration to use only HTTPS:

1. Make sure a server certificate and associated private key are available:

```
$ keytool \
 -list \
 -alias server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12
...
server-cert, <date>, PrivateKeyEntry
...
```

2. Disable the HTTP connection handler to prevent (cleartext) HTTP access:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name HTTP \
 --set enabled:false \
 --trustAll \
 --no-prompt
```

3. Configure the HTTP connection handler to use HTTPS access.

   The following example shows how to set the port to 8443 and perform TLS using the default server certificate:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name HTTP \
 --set enabled:true \
 --set listen-port:8443 \
 --set use-ssl:true \
 --set key-manager-provider:"Default Key Manager" \
 --set trust-manager-provider:"JVM Trust Manager" \
 --trustAll \
 --no-prompt
```

4. (Optional) Enable the HTTP access log.

   - The following command enables JSON-based HTTP access logging as described in "Configuring JSON Access Logs":

   ```
   $ dsconfig \
    set-log-publisher-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --publisher-name "Json File-Based HTTP Access Logger" \
    --set enabled:true \
    --no-prompt \
    --trustAll
   ```

   - The following command enables HTTP access logging as described in "Standard HTTP Access Logs":

   ```
   $ dsconfig \
    set-log-publisher-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --publisher-name "File-Based HTTP Access Logger" \
    --set enabled:true \
    --no-prompt \
    --trustAll
   ```

5. (Optional) If the deployment requires SSL client authentication, set the properties `ssl-client-auth-policy` and `trust-manager-provider` appropriately.

6. After you set up the HTTP connection handler, make sure that at least one HTTP endpoint configuration object is enabled.

   For details, see "To Set Up REST Access to User Data" or "To Set Up REST Access to Administrative Data".

To Set Up REST Access to User Data

The way directory data appears to client applications is configurable. You can configure one or more Rest2ldap endpoints to expose user directory data over HTTP. The mapping defined for the Rest2ldap endpoint defines a mapping between JSON resources and LDAP entries. The mapping is expressed in a configuration file, by default `/path/to/opendj/config/rest2ldap/endpoints/api/example-v1.json`. The configuration is described in "*REST to LDAP Configuration*" in the *Reference*.

The default Rest2ldap endpoint exposes the RESTful API demonstrated in "*Performing RESTful Operations*" in the *Developer's Guide*. The default mapping works with generated sample data, and Example.com data imported in evaluation mode.

If you set up the directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*, you can skip to Step 3:

1. (Optional) If necessary, change the properties of the default Rest2ldap endpoint, or create a new endpoint.

   A Rest2ldap HTTP endpoint named `/api` after its `base-path` is provided by default. The `base-path` must be the same as the name, and is read-only after creation. By default, the `/api` endpoint requires authentication.

   The following example enables the `/api` endpoint using the default mapping and HTTP Basic authorization. Adjust these settings as necessary:

   ```
   $ dsconfig \
    set-http-endpoint-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --endpoint-name /api \
    --set authorization-mechanism:"HTTP Basic" \
    --set config-directory:config/rest2ldap/endpoints/api \
    --set enabled:true \
    --no-prompt \
    --trustAll
   ```

   Alternatively, you can create another Rest2ldap endpoint to expose a different view of the directory data, or to publish data under an alternative base path, such as `/rest`:

   ```
   $ dsconfig \
    create-http-endpoint \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --endpoint-name /rest \
    --type rest2ldap-endpoint \
    --set authorization-mechanism:"HTTP Basic" \
    --set config-directory:config/rest2ldap/endpoints/api \
    --set enabled:true \
    --no-prompt \
    --trustAll
   ```

2. (Optional) If necessary, adjust the endpoint configuration to use an alternative HTTP authorization mechanism.

By default, the Rest2ldap endpoint maps HTTP Basic authentication to LDAP authentication to set the authorization identity for operations. You can change the `authorization-mechanism` setting to use a different HTTP authorization mechanism as described in "To Set Up HTTP Authorization".

3. (Optional) Try reading a resource.

   The following example demonstrates reading the resource that corresponds to Barbara Jensen's entry as a JSON resource:

```
$ curl http://bjensen:hifalutin@opendj.example.com:8080/api/users/bjensen?_prettyPrint=true
{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : { },
  "userName" : "bjensen@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "bjensen@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  }
}
```

4. (Optional) If the HTTP connection handler is configured to use HTTPS, try reading an entry over HTTPS.

   The following example writes the (self-signed) server certificate into a trust store file, and uses the file to trust the server when setting up the HTTPS connection:

```
$ keytool \
 -export \
 -rfc \
 -alias server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12 \
 -file server-cert.pem
Certificate stored in file <server-cert.pem>

$ curl \
 --cacert server-cert.pem \
```

```
 --user bjensen:hifalutin \
 https://opendj.example.com:8443/api/users/bjensen?_prettyPrint=true
{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : { },
  "userName" : "bjensen@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "bjensen@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  }
}
```

Notice the `--cacert server-cert.pem` option used with the **curl** command. This is the way to specify a self-signed server certificate when using HTTPS.

To Set Up HTTP Authorization

HTTP authorization mechanisms define how DS servers authorize client HTTP requests to directory data. Authorization mechanisms map credentials from an HTTP-based protocol, such as HTTP Basic authentication or OAuth 2.0, to LDAP credentials.

Multiple HTTP authorization mechanisms can be enabled simultaneously, and assigned to HTTP endpoints, such as Rest2ldap endpoints described in "To Set Up REST Access to User Data" or the Admin endpoint described in "To Set Up REST Access to Administrative Data".

By default, these HTTP authorization mechanisms are supported:

HTTP Anonymous

Handle anonymous HTTP requests, optionally binding with a specified DN.

If no bind DN is specified (default), anonymous LDAP requests are used.

This mechanism is enabled by default unless the server was installed with the **setup** command option, `--productionMode`.

HTTP Basic (enabled by default)

Handle HTTP Basic authentication requests by mapping the HTTP Basic identity to a user's directory account for the underlying LDAP operation.

By default, the Exact Match identity mapper with its default configuration is used to map the HTTP Basic user name to an LDAP `uid`. The DS server then searches in all local public naming contexts to find the user's entry based in the `uid` value.

HTTP OAuth2 CTS

Handle OAuth 2.0 requests as an OAuth 2.0 resource server, where a directory service acts as an AM Core Token Service (CTS) store.

When the client bearing an OAuth 2.0 access token presents the token to access the JSON resource, the server tries to resolve the access token against the CTS data that it serves for AM. If the access token resolves correctly (is found in the CTS data and has not expired), the DS server extracts the user identity and OAuth 2.0 scopes. If the required scopes are present and the token is valid, it maps the user identity to a user's directory account for the underlying LDAP operation.

This mechanism makes it possible to resolve access tokens by making an internal request, avoiding a request to AM. *This mechanism does not, however, ensure that the token requested will have already been replicated to the directory server where the request is routed.*

AM's CTS store is constrained to a specific layout. The `authzid-json-pointer` must therefore use `userName/0` for the user identifier.

HTTP OAuth2 OpenAM

Handle OAuth 2.0 requests as an OAuth 2.0 resource server, where the directory service sends requests to AM for access token resolution.

When the client bearing an OAuth 2.0 access token presents the token to access the JSON resource, the directory service requests token information from AM. If the access token is valid, the DS server extracts the user identity and OAuth 2.0 scopes. If the required scopes are present, it maps the user identity to a user's directory account for the underlying LDAP operation.

As access token resolution requests ought to be sent over HTTPS, you can configure a trust store manager if necessary to trust the authorization server certificate, and a key store manager to obtain the DS server certificate if the authorization server requires mutual authentication.

HTTP OAuth2 Token Introspection (RFC7662)

Handle OAuth 2.0 requests as an OAuth 2.0 resource server, where the DS server sends requests to an RFC 7662-compliant authorization server for access token resolution.

RFC 7662, *OAuth 2.0 Token Introspection*, defines a standard method for resolving access tokens. The DS server must be registered as a client of the authorization server.

When the client bearing an OAuth 2.0 access token presents the token to access the JSON resource, the DS server requests token introspection from the authorization server. If the access

token is valid, the DS server extracts the user identity and OAuth 2.0 scopes. If the required scopes are present, it maps the user identity to a user's directory account for the underlying LDAP operation.

As access token resolution requests ought to be sent over HTTPS, you can configure a trust store manager if necessary to trust the authorization server certificate, and a key store manager to obtain the DS server certificate if the authorization server requires mutual authentication.

Note

The HTTP OAuth2 File mechanism is an internal interface intended for testing and not supported for production use.

When more than one authentication mechanism is specified, mechanisms are applied in the following order:

- If the client request has an `Authorization` header, and an OAuth 2.0 mechanism is specified, the server attempts to apply the OAuth 2.0 mechanism.
- If the client request has an `Authorization` header, or has the custom credentials headers specified in the configuration, and an HTTP Basic mechanism is specified, the server attempts to apply the Basic Auth mechanism.
- Otherwise, if an HTTP anonymous mechanism is specified, and none of the previous mechanisms apply, the server attempts to apply the mechanism for anonymous HTTP requests.

There are many possibilities when configuring HTTP authorization mechanisms. *This procedure shows only one OAuth 2.0 example.*

The example that follows demonstrates a server configured for tests (insecure connections) to request OAuth 2.0 token information from AM. It uses settings as listed in "Settings for OAuth 2.0 Example With AM".

Download ForgeRock Access Management software from the ForgeRock BackStage download site.

Settings for OAuth 2.0 Example With AM

Setting	Value
OpenAM URL	`http://openam.example.com:8088/openam`
Authorization server endpoint	`/oauth2/tokeninfo` (top-level realm)
Identity repository	`opendj.example.com:1389` installed for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*
OAuth 2.0 client ID	`myClientID`
OAuth 2.0 client secret	`password`
OAuth 2.0 client scopes	`read`, `uid`, `write`

Setting	Value
Rest2ldap configuration	Default settings. See "To Set Up REST Access to User Data".

Read the ForgeRock Access Management documentation if necessary to install and configure AM. Then follow these steps to try the demonstration:

1. Update the default HTTP OAuth2 OpenAM configuration:

```
$ dsconfig \
 set-http-authorization-mechanism-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mechanism-name "HTTP OAuth2 OpenAM" \
 --set enabled:true \
 --set token-info-url:http://openam.example.com:8088/openam/oauth2/tokeninfo \
 --no-prompt \
 --trustAll
```

2. Update the default Rest2ldap endpoint configuration to use HTTP OAuth2 OpenAM as the authorization mechanism:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set authorization-mechanism:"HTTP OAuth2 OpenAM" \
 --no-prompt \
 --trustAll
```

3. Obtain an access token with the appropriate scopes:

```
$ curl \
 --request POST \
 --user "myClientID:password" \
 --data "grant_type=password&username=bjensen&password=hifalutin&scope=read%20uid%20write" \
 http://openam.example.com:8088/openam/oauth2/access_token
{
 "access_token": "token-string",
 "scope": "uid read write",
 "token_type": "Bearer",
 "expires_in": 3599
}
```

   In production systems, make sure you use HTTPS when obtaining access tokens.

4. Request a resource at the Rest2ldap endpoint using HTTP Bearer authentication with the access token:

```
$ curl \
 --header "Authorization: Bearer token-string" \
 http://opendj.example.com:8080/api/users/bjensen?_prettyPrint=true
{
  "_id": "bjensen",
  "_rev": "<revision>",
  "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta": {},
  "userName": "bjensen@example.com",
  "displayName": ["Barbara Jensen", "Babs Jensen"],
  "name": {
    "givenName": "Barbara",
    "familyName": "Jensen"
  },
  "description": "Original description",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1862",
    "emailAddress": "bjensen@example.com"
  },
  "uidNumber": 1076,
  "gidNumber": 1000,
  "homeDirectory": "/home/bjensen",
  "manager": {
    "_id": "trigden",
    "displayName": "Torrey Rigden"
  }
}
```

In production systems, make sure you use HTTPS when presenting access tokens.

To Set Up REST Access to Administrative Data

The APIs for configuring and monitoring DS servers are under the following endpoints:

`/admin/config`

Provides a REST API to the server configuration with a JSON-based view of `cn=config` and the configuration backend.

By default, this endpoint is protected by the HTTP Basic authorization mechanism. Users reading and editing the configuration must have appropriate privileges, such as `config-read` and `config-write`, as described below.

Each LDAP entry maps to a resource under `/admin/config`, with default values shown in the resource even if they are not set in the LDAP representation.

`/alive`

Provides an endpoint to check whether the server is currently *alive*, meaning that its internal checks have not found any errors that would require administrative action.

By default, this endpoint returns a status code to anonymous requests, and also supports authenticated requests. For details, see "Monitoring Liveness over HTTP".

`/healthy`

Provides an endpoint to check whether the server is currently *healthy*, meaning that it is alive and any replication delays are below a configurable threshold.

By default, this endpoint returns a status code to anonymous requests, and also supports authenticated requests. For details, see "Monitoring Ability to Handle Requests Over HTTP".

`/metrics/api`

Provides a REST API to the server monitoring information with a read-only JSON-based view of `cn=monitor` and the monitoring backend.

By default, this endpoint is protected by the HTTP Basic authorization mechanism. Users reading monitoring information must have the `monitor-read` privilege.

Each LDAP entry maps to a resource under `/metrics/api`.

`/metrics/prometheus`

Provides an API to the server monitoring information for use with Prometheus monitoring software.

By default, this endpoint is protected by the HTTP Basic authorization mechanism. Users reading monitoring information must have the `monitor-read` privilege.

To use the Admin endpoint APIs, follow these steps:

1. Grant users access to the endpoints as appropriate:

   - For access to `/admin/config`, assign `config-read` or `config-write` privileges.

     The following example assigns the `config-read` privilege to Kirsten Vaughan:

     ```
     $ cat config-read.ldif
     dn: uid=kvaughan,ou=People,dc=example,dc=com
     changetype: modify
     add: ds-privilege-name
     ds-privilege-name: config-read

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

   - For access to `/metrics` endpoints, assign the `monitor-read` privilege.

The following example adds the `monitor-read` privilege to Kirsten Vaughan's entry:

```
$ cat monitor-read.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: monitor-read

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

As an alternative, you can add a monitoring account when setting up the server. For an example, see "To Set Up a Directory Server" in the *Installation Guide*.

For more detail, see "Configuring Privileges".

2. (Optional) If necessary, adjust the `authorization-mechanism` setting for the Admin endpoint.

   By default, the Admin endpoint uses the HTTP Basic authorization mechanism. The HTTP Basic authorization mechanism default configuration resolves the user identity extracted from the HTTP request to an LDAP user identity as follows:

   1. If the request has an `Authorization: Basic` header for HTTP Basic authentication, the server extracts the username and password.

   2. If the request has `X-OpenIDM-Username` and `X-OpenIDM-Password` headers, the server extracts the username and password.

   3. The server uses the default Exact Match identity mapper to search for a unique match between the username and the UID attribute value of an entry in the local public naming contexts of the DS server.

      In other words, in LDAP terms, it searches under all user data base DNs for `(uid=`*`http-username`*`)`. The username `kvaughan` maps to the example entry with DN `uid=kvaughan,ou=People,dc=example,dc=com`.

   For details on configuring HTTP authorization mechanisms, see "To Set Up HTTP Authorization".

3. (Optional) Consider protecting traffic to the Admin endpoint by using HTTPS as described in "To Set Up HTTP Access".

4. Test access to the endpoint as an authorized user.

   The examples below use the (self-signed) server certificate which the following command writes into file named `server-cert.pem`:

```
$ keytool \
 -export \
 -rfc \
 -alias server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12 \
 -file server-cert.pem
Certificate stored in file <server-cert.pem>
```

The following example demonstrates reading the Admin endpoint resource under `/admin/config`:

```
$ curl \
 --cacert server-cert.pem \
 --user kvaughan:bribery \
 https://opendj.example.com:8443/admin/config/http-endpoints/%2Fadmin?_prettyPrint=true
{
  "_id" : "/admin",
  "_rev" : "<revision>",
  "_schema" : "admin-endpoint",
  "java-class" : "org.opends.server.protocols.http.rest2ldap.AdminEndpoint",
  "base-path" : "/admin",
  "enabled" : true,
  "authorization-mechanism" : "HTTP Basic"
}
```

Notice how the path to the resource in the example above, `/admin/config/http-endpoints/%2Fadmin`, corresponds to the DN of the entry under cn=config, which is `ds-cfg-base-path=/admin,cn=HTTP Endpoints,cn=config`.

The following example demonstrates reading everything under `/metrics/api`:

```
$ curl \
 --cacert server-cert.pem \
 --user kvaughan:bribery \
 https://opendj.example.com:8443/metrics/api?_queryFilter=true
```

To Set Up REST to LDAP Gateway

Follow these steps to set up the REST to LDAP gateway Servlet to access your directory service.

1. Download and install the gateway as described in "To Install the REST to LDAP Gateway" in the *Installation Guide*.

2. Adjust the configuration for your directory service as described in "*REST to LDAP Configuration*" in the *Reference*.

DSML Client Access

Directory Services Markup Language (DSML) client access is implemented as a servlet that runs in a web application container.

You configure DSML client access by editing the `WEB-INF/web.xml` after you deploy the web application. In particular, you must at least set the `ldap.host` and `ldap.port` parameters if they differ from the default values, which are `localhost` and `389`.

The list of DSML configuration parameters, including those that are optional, consists of the following:

`ldap.host`

Required parameter indicating the host name of the underlying directory service. Default: `localhost`.

`ldap.port`

Required parameter indicating the LDAP port of the underlying directory service. Default: 389.

`ldap.userdn`

Optional parameter specifying the DN used by the DSML gateway to bind to the underlying directory service. Not used by default.

`ldap.userpassword`

Optional parameter specifying the password used by the DSML gateway to bind to the underlying directory service. Not used by default.

`ldap.authzidtypeisid`

This parameter can help you set up the DSML gateway to do HTTP Basic Access Authentication, given the appropriate mapping between the user ID, and the user's entry in the directory.

Required boolean parameter specifying whether the HTTP Authorization header field's Basic credentials in the request hold a plain ID, rather than a DN. If set to `true`, then the gateway performs an LDAP SASL bind using SASL plain, enabled by default in DS servers to look for an exact match between a `uid` value and the plain ID value from the header. In other words, if the plain ID is `bjensen`, and that corresponds in the directory service to Babs Jensen's entry with DN `uid=bjensen,ou=people,dc=example,dc=com`, then the bind happens as Babs Jensen. Note also that you can configure DS identity mappers for scenarios that use a different attribute than `uid`, such as the `mail` attribute.

Default: `false`

`ldap.usessl`

Required parameter indicating whether `ldap.port` points to a port listening for LDAPS (LDAP/SSL) traffic. Default: `false`.

`ldap.usestarttls`

Required parameter indicating whether to use StartTLS to connect to the specified `ldap.port`. Default: `false`.

`ldap.trustall`

Required parameter indicating whether to blindly trust all certificates presented to the DSML gateway when using secure connections (LDAPS or StartTLS). Default: `false`.

`ldap.truststore.path`

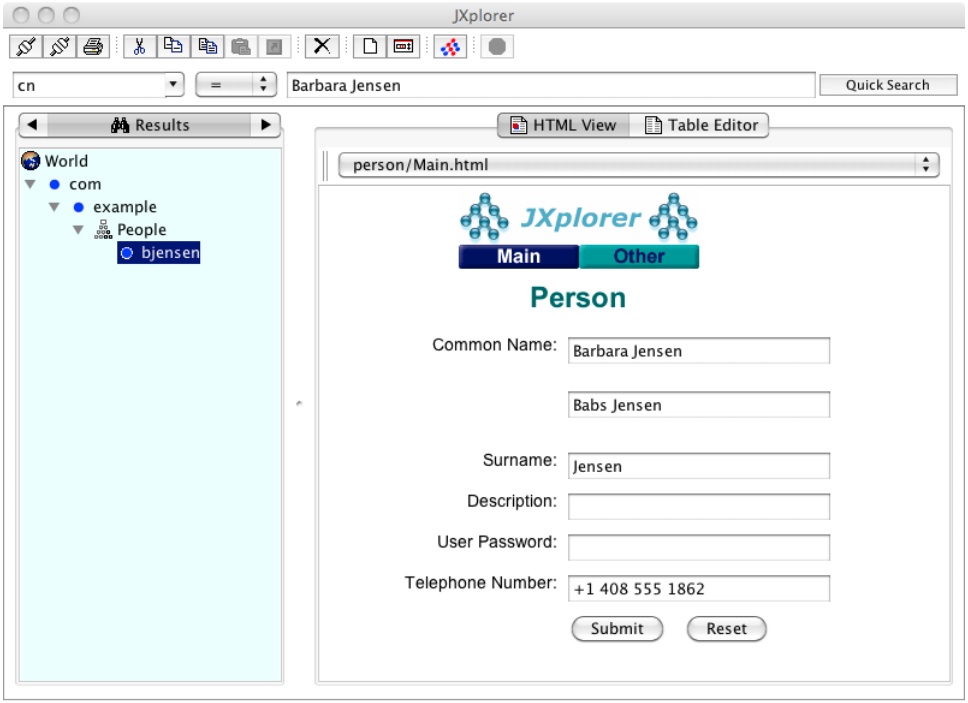
Optional parameter indicating the truststore used to verify certificates when using secure connections. If you want to connect using LDAPS or StartTLS, and do not want the gateway blindly to trust all certificates, then you must set up a truststore. Not used by default.

`ldap.truststore.password`

Optional parameter indicating the truststore password. If you set up and configure a truststore, then you need to set this as well. Not used by default.

The DSML servlet translates between DSML and LDAP, and passes requests to the directory service. For initial testing purposes, you might try JXplorer, where DSML Service: */webapp-dir*/DSMLServlet. Here, *webapp-dir* refers to the name of the directory in which you unpacked the DSML `.war`. "JXplorer Accessing the Directory Service" shows the result.

JXplorer Accessing the Directory Service



JMX Client Access

You configure Java Management Extensions (JMX) client access by using the command-line tool, **dsconfig**.

To Set Up JMX Access

1. Set server Java arguments appropriately to avoid regular full garbage collection (GC) events.

   JMX is based on Java Remote Method Invocation (RMI), which uses references to objects. By default, the JMX client and server perform a full GC periodically to clean up stale references. As a result, the default settings cause JMX to cause a full GC every hour.

   To prevent hourly full GCs when using JMX, add the `-XX:+DisableExplicitGC` option to the list of `start-ds.java-args` arguments. You can do this by editing the `config/java.properties` file and restarting the server.

Avoid using this argument when importing LDIF online using the **import-ldif** command. The import process uses garbage collection to work around memory management issues.

2. Configure the server to activate JMX access.

   The following example uses the reserved port number, 1689:

```
$ dsconfig \
 create-connection-handler \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name JMX \
 --type jmx \
 --set enabled:true \
 --set listen-port:1689 \
 --trustAll \
 --no-prompt
```

   The change takes effect immediately.

To Configure Access To JMX

After you set up a server to listen for JMX connections, you must assign privileges in order to allow a user to connect over JMX:

1. Assign the necessary privileges to the account that connects over JMX.

   The following privileges apply:

   - `jmx-notify`
   - `jmx-read`
   - `jmx-write`
   - `monitor-read`

   The following LDIF modification assigns these privileges to the application account with DN `cn=My App,ou=Apps,dc=example,dc=com`:

```
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: jmx-notify
ds-privilege-name: jmx-read
ds-privilege-name: jmx-write
ds-privilege-name: monitor-read
```

For details, see "Configuring Privileges".

2. Connect using the service URI, user name, and password:

Service URI

Full URI to the service including the hostname or IP address and port number for JMX where the DS server listens for connections.

For example, if the server hostname is `opendj.example.com`, and the DS server listens for JMX connections on port 1689, then the service URI is `service:jmx:rmi:///jndi/rmi://opendj.example.com:1689/org.opends.server.protocols.jmx.client-unknown`.

User name

The full DN of the user with privileges to connect over JMX, such as `cn=My App,ou=Apps,dc=example,dc=com`.

Password

The bind password for the user.

LDIF File Access

The LDIF connection handler lets you make changes to directory data by placing LDIF files in a file system directory that the DS server regularly polls for changes. The LDIF, once consumed, is deleted.

To Set Up LDIF File Access

1. Add the directory where you put LDIF to be processed:

```
$ mkdir /path/to/opendj/config/auto-process-ldif
```

This example uses the default value of the `ldif-directory` property for the LDIF connection handler.

2. Activate LDIF file access:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDIF \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

The change takes effect immediately.

SNMP Access

For instructions on setting up the SNMP connection handler, see "SNMP-Based Monitoring".

Chapter 6
Configuring Privileges and Access Control

DS software supports these mechanisms to protect access:

- *Access control instructions* (ACI)

  Access control instructions apply to directory data, providing fine-grained control over what a user or group member is authorized to do in terms of LDAP operations. Most access control instructions specify scopes (targets) such that an administrative user who has all access to `dc=example,dc=com` need not have any access to `dc=example,dc=org`.

- Administrative *privileges*

  Privileges control the administrative tasks that users can perform, such as bypassing the access control mechanism, performing backup and restore operations, making changes to the configuration, and other tasks.

  By default, privileges restrict administrative access to directory root users, though any user can be assigned a privilege. Privileges apply to DS servers, and do not have a scope.

- *Global access control policies*

  Global access control policies provide coarse-grained access control suitable for use on proxy servers, where the lack of local access to directory data makes ACIs a poor fit.

Note

Access control instructions (ACI) and global access control policies rely on different access control handlers, which implement different access control models. ACIs rely on the DSEE-compatible access control handler. (DSEE refers to Sun Java System Directory Server Enterprise Edition.) Global access control policies rely on the policy-based access control handler. A server can only use one handler at a time.

Take the following constraints into consideration:

- When the policy-based handler is configured, ACIs have no effect.
- When the DSEE-compatible handler is configured, global access control policies have no effect.
- When a server is set up as a directory server, it uses the DSEE-compatible access control handler, with ACIs in directory data and global ACIs in the configuration.
- When a server is set up as a directory proxy server, it uses the policy-based access control policy handler, and global access control policies.

- Once the server has been set up, the choice of access control handler cannot be changed with the **dsconfig** command.

This chapter covers each mechanism. In this chapter you will learn to:

- Configure privileges for directory administration
- Configure coarse-grained global access control policies
- Read and write access control instructions
- Configure access rights by setting access control instructions
- Evaluate effective access rights for a particular user

Some operations require both privileges and access control. For example, in order to reset user's passwords, an administrator needs the `password-reset` privilege and access to write `userPassword` values on user entries. By combining access control and privileges, you can effectively restrict scope of privileges to a particular branch of the Directory Information Tree.

About Privileges

Privileges provide access control for server administration independently from ACIs.

The default directory superuser, `cn=Directory Manager`, is granted the privileges marked with an asterisk (*). Other administrator users can be assigned these privileges, too:

`backend-backup`*

Request a task to back up data

`backend-restore`*

Request a task to restore data from backup

`bypass-acl`*

Perform operations without regard to ACIs

`bypass-lockdown`*

Perform operations without regard to lockdown mode

`cancel-request`*

Cancel any client request

`changelog-read`*

Read the changelog (under `cn=changelog`)

`config-read`*

Read the server configuration

`config-write`*

Change the server configuration

`data-sync`

Perform data synchronization

`disconnect-client`*

Close any client connection

`jmx-notify`

Subscribe to JMX notifications

`jmx-read`

Read JMX attribute values

`jmx-write`

Write JMX attribute values

`ldif-export`*

Export data to LDIF

`ldif-import`*

Import data from LDIF

`modify-acl`*

Change ACIs

`monitor-read`*

Read metrics under `cn=monitor`, `/metrics/api`, `/metrics/prometheus`, and over JMX

`password-reset`*

Reset other users' passwords

`privilege-change`*

Change the privileges assigned to users

Take great care when assigning this privilege, as it allows the user to assign themselves all other administrative privileges.

`proxied-auth`

Use the Proxied Authorization control

`server-lockdown`*

Put the server into and take the server out of lockdown mode

`server-restart`*

Request a task to restart the server

`server-shutdown`*

Request a task to stop the server

`subentry-write`*

Perform LDAP subentry write operations

`unindexed-search`*

Search using a filter with no corresponding index

`update-schema`*

Change LDAP schema definitions

Configuring Privileges

For all directory administrators, you can change privileges with the **ldapmodify** command as shown in the following procedures:

- "To Add Privileges for an Individual Administrator"
- "To Add Privileges for a Group of Administrators"
- "To Limit Inherited Privileges"

To Add Privileges for an Individual Administrator

Privileges are specified using the `ds-privilege-name` operational attribute, which you can change using the **ldapmodify** command.

1. Determine the privileges to add:

```
$ cat privileges.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: config-read
ds-privilege-name: password-reset
```

This example lets the user read the server configuration, and reset user passwords. In order for the user to be able to change a user password, you must also allow the modification using ACIs.

For this example, Kirsten Vaughan is a member of the Directory Administrators group for Example.com, and already has access to modify user entries.

Prior to having the privileges, Kirsten gets messages about insufficient access when trying to read the server configuration, or trying to reset a user password:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN cn=config \
 "(objectclass=*)"
# The LDAP search request failed: 50 (Insufficient Access Rights)
# Additional Information:  You do not have sufficient privileges to perform search operations in the
 Directory Server configuration
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --authzID "dn:uid=scarter,ou=People,dc=example,dc=com" \
 --newPassword changeit
The LDAP password modify operation failed: 50 (Insufficient Access Rights)
Additional Information:  You do not have sufficient privileges to perform
password reset operations
```

2. Apply the change as a user with the `privilege-change` privilege:

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

At this point, Kirsten can perform the operations requiring privileges:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN cn=config \
 "(objectclass=*)"
dn: cn=config
…
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --authzID "dn:uid=scarter,ou=People,dc=example,dc=com" \
 --newPassword changeit
The LDAP password modify operation was successful
```

To Add Privileges for a Group of Administrators

For deployments with more than one administrator, use groups to define administrative rights.

You can use a collective attribute subentry to specify privileges for the administrator group.

Collective attributes provide a standard mechanism for defining attributes that appear on all the entries in a particular subtree. For details, see "Collective Attributes" in the *Developer's Guide*. DS servers extend collective attributes to give you fine-grained control over which entries are in scope. In addition, DS servers let you use virtual attributes, such as `isMemberOf` to construct the filter for targeting entries in scope. This allows you, for example, to define administrative privileges that apply to all users who belong to an administrator group:

1. Create an LDAP subentry that specifies the collective attributes:

```
$ cat collective.ldif
dn: cn=Administrator Privileges,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Administrator Privileges
ds-privilege-name;collective: config-read
ds-privilege-name;collective: config-write
ds-privilege-name;collective: ldif-export
ds-privilege-name;collective: modify-acl
ds-privilege-name;collective: password-reset
ds-privilege-name;collective: proxied-auth
subtreeSpecification: {base "ou=people", specificationFilter
  "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)" }

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

The Directory Administrators group for Example.com includes members like Harry Miller.

The `base` entry identifies the branch that holds administrator entries. For details on how subentries apply, see "Understanding Subentry Scope".

2. Observe that the change takes effect immediately:

```
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "uid=hmiller,ou=people,dc=example,dc=com" \
 --bindPassword hillock \
 --authzID "dn:uid=scarter,ou=People,dc=example,dc=com"
The LDAP password modify operation was successful
Generated Password:  <password>
```

To Limit Inherited Privileges

When privileges are set as described in "To Add Privileges for a Group of Administrators", the same list of privileges is applied to every target account.

In some cases, the list of inherited privileges can be too broad. DS servers have a mechanism to limit effective privileges by preceding the privilege attribute value with a `-`.

The following steps show how to prevent Kirsten Vaughan from resetting passwords when the privilege is assigned as described in "To Add Privileges for a Group of Administrators":

1. Check the privilege settings for the account:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 ds-privilege-name
dn: uid=kvaughan,ou=People,dc=example,dc=com
ds-privilege-name: config-read
ds-privilege-name: password-reset
```

2. Set the privilege attribute for the account to deny the privilege:

```
$ cat restrict-privileges.ldif
dn: uid=kvaughan,ou=people,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: -password-reset

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

3. Observe that the privilege is no longer in effect:

```
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --authzID "dn:uid=scarter,ou=People,dc=example,dc=com"
The LDAP password modify operation failed: 50 (Insufficient Access Rights)
Additional Information:  You do not have sufficient privileges to perform
password reset operations
```

About ACIs

DS server ACIs exist as operational `aci` attribute values on directory entries, and as global ACI attributes stored in the configuration. ACIs apply to a scope defined in the instruction. They set permissions that depend on what operation is requested, who requested the operation, and how the client connected to the server.

For example, the ACIs on the following entry allow anonymous read access to all attributes except passwords, and allow read-write access for directory administrators under `dc=example,dc=com`:

```
dn: dc=example,dc=com
objectClass: domain
objectClass: top
dc: example
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr != "userPassword")(version 3.0;acl "Anonymous read-search access";
 allow(read, search, compare)(userdn = "ldap:///anyone");)
aci: (target="ldap:///dc=example,dc=com")
 (targetattr = "*")(version 3.0; acl "allow all Admin group";
 allow(all) groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)
```

The DS server default behavior is that no access is allowed unless it is specifically granted by an ACI. (The `bypass-acl` privilege, assigned to certain users such as `cn=Directory Manager`, can allow users to bypass access control checks.)

DS servers provide default global ACIs to facilitate evaluation while maintaining a reasonable security policy. By default, users are granted access to:

- Read the root DSE
- Read the LDAP schema
- Use certain LDAP controls and extended operations
- Modify their own entries
- Read public data and operational attributes

Global ACIs are defined on the access control handler, and apply to the entire server. You must adjust the default global ACIs to match the security policies for your organization, for example, to restrict anonymous access.

ACI attribute values use a specific language described in this section. Although ACI attribute values can become difficult to read in LDIF, the basic syntax is simple:

```
targets(version 3.0;acl "name";permissions subjects;)
```

The following list briefly explains the variables in the syntax above:

targets

The *targets* specifies entries, attributes, controls, and extended operations to which the ACI applies.

To include multiple *targets*, enclose each individual target in parentheses, `()`. When you specify multiple targets, all targets must match for the ACI to apply (`AND`).

name

Supplies a human-readable description of what the ACI does.

permissions

Defines which actions to allow, and which to deny. Paired with *subjects*.

subjects

Identifies clients to which the ACI applies depending on who connected, and when, where, and how they connected. Paired with *permissions*.

Separate multiple pairs of *permissions-subjects* definitions with semicolons, `;`. When you specify multiple permissions-subjects pairs, at least one must match (`OR`).

ACI Targets

The seven types of ACI targets identify the objects to which the ACI applies. Most expressions allow you to use either `=` to specify that the target should match the value or `!=` to specify that the target should not match the value:

`(target [!]= "ldap:///DN")`

Sets the scope to the entry with distinguished name *DN*, and to child entries.

You can use asterisks, `*`, to replace attribute types, attribute values, and entire DN components.

In other words, the following specification targets both `uid=bjensen,ou=People,dc=example,dc=com` and `cn=My App,ou=Apps,dc=example,dc=com`:

```
(target = "ldap:///*=*,*,dc=example,dc=com")
```

The *DN* must be in the subtree of the entry where the ACI is defined.

If you do not specify `target`, then the entry holding this ACI is affected. If `targetscope` is also omitted, then this entry and all subordinates are affected.

`(targetattr [!]= "attr-list")`

Replace *attr-list* with a list of attribute type names, such as `userPassword`, separating multiple attribute type names with `||`.

This specification affects the entry where the ACI is located, or the entries specified by other targets in the ACI.

You can use an asterisk, `*`, to specify all user attributes, although you will see better performance when explicitly including or excluding attribute types as needed. You can use a plus sign, `+`, to specify all operational attributes.

A negated *attr-list* of operational attributes will only match other operational attributes and never any user attributes, and vice-versa.

If you do not include this target specification, then by default no attributes are affected by the ACI.

`(targetfilter [!]= "ldap-filter")`

Sets the scope to match the *ldap-filter* dynamically, as in an LDAP search. The *ldap-filter* can be any valid LDAP filter.

`(targattrfilters = "expression")`

Use this target specification when managing changes made to particular attributes.

The *expression* takes one of the following forms. Separate expressions with commas (`,`):

```
op=attr1:filter1[&& attr2:filter2 …][,op=attr3:filter3[&& attr4:filter4 …] …]
```

The *op* can be either `add` for operations creating attributes, or `del` for operations removing them.

Replace *attr* with an attribute type. Replace *filter* with an LDAP filter that corresponds to the *attr* attribute type.

`(targetscope = "base|onelevel|subtree|subordinate")`

- `base` refers to the entry with the ACI.
- `onelevel` refers to immediate children.
- `subtree` refers to the base entry and all children.
- `subordinate` refers to all children only.

If you do not specify `targetscope`, the default is `subtree`.

`(targetcontrol [!]= "OID")`

Replace *OID* with the object identifier for the LDAP control to target. Separate multiple OIDs with `||`.

To use an LDAP control, the bind DN user must have `allow(read)` permissions.

This target cannot be restricted to a specific subtree by combining it with another target.

`(extop [!]= "OID")`

Replace *OID* with the object identifier for the extended operation to target. Separate multiple OIDs with `||`.

To use an LDAP extended operation, the bind DN user must have `allow(read)` permissions.

This target cannot be restricted to a specific subtree by combining it with another target.

ACI Permissions

ACI permission definitions take one of the following forms:

```
allow(action[, action …])
```

```
deny(action[, action …])
```

Tip

Although `deny` is supported, avoid restricting permissions by using `deny`. Instead, explicitly `allow` access only as needed. What looks harmless and simple in tests and examples can grow difficult to maintain in a real-world deployment with nested ACIs.

Replace *action* with one of the following:

`add`

Entry creation, as for an LDAP add operation.

`all`

All permissions, except `export`, `import`, `proxy`.

`compare`

Attribute value comparison, as for an LDAP compare operation.

`delete`

Entry deletion, as for an LDAP delete operation.

`export`

Entry export during a modify DN operation.

Despite the name, this action is unrelated to LDIF export operations.

`import`

Entry import during a modify DN operation.

Despite the name, this action is unrelated to LDIF import operations.

`proxy`

Access the ACI target using the rights of another user.

`read`

Read entries and attributes, or use an LDAP control or extended operation.

`search`

Search the ACI targets.

Combine with `read` to read the search results.

`selfwrite`

Add or delete own DN from a group.

`write`

Modify attributes on ACI target entries.

ACI Subjects

ACI subjects match characteristics of the client connection to the server. Use subjects to restrict whether the ACI applies depending on who connected, and when, where, and how they connected. Most expressions allow you to use either `=` to specify that the condition should match the value or `!=` to specify that the condition should not match the value:

`authmethod [!]= "none|simple|ssl|sasl mech"`

- `none` means do not check.
- `simple` means simple authentication.
- `ssl` refers to certificate-based authentication over LDAPS.
- `sasl mech` refers to SASL, where *mech* is DIGEST-MD5, EXTERNAL, or GSSAPI.

`dayofweek [!]= "day[, day …]"`

Replace *day* with one of:

```
sun
mon
tue
wed
thu
fri
sat
```

`dns [!]= "hostname"`

Use asterisks, `*`, to replace name components, such as `dns = "*.example.com"`.

`groupdn [!]= "ldap:///DN[|| ldap:///DN …]"`

Replace *DN* with the distinguished name of a group to permit or restrict access for members.

```
ip [!]= "addresses"
```

The *addresses* can be specified for IPv4 or IPv6.

IPv6 addresses are specified in brackets as `ldap://[address]/subnet-prefix` where */subnet-prefix* is optional.

You can specify:

- Individual IPv4 addresses
- Addresses with asterisks (`*`) to replace subnets and host numbers
- CIDR notation
- Forms such as `192.168.0.*+255.255.255.0` to specify subnet masks

```
ssf = "strength"
ssf != "strength"
ssf > "strength"
ssf >= "strength"
ssf < "strength"
ssf <= "strength"
```

The security strength factor (`ssf`) pertains to the cipher key strength for connections using DIGEST-MD5, GSSAPI, SSL, or TLS.

For example, to require that the connection must have a cipher strength of at least 256 bits, specify `ssf >= "256"`.

```
timeofday = "hhmm"
timeofday != "hhmm"
timeofday > "hhmm"
timeofday >= "hhmm"
timeofday < "hhmm"
timeofday <= "hhmm"
```

The *hhmm* is expressed as on a 24-hour clock.

For example, 1:15 PM is written `1315`.

```
userattr [!]= "attr#value"
userattr [!]= ldap-url#LDAPURL"
userattr [!]= "[parent[child-level]. ]attr#GROUPDN|USERDN"
```

The `userattr` subject specifies an attribute that must match on both the bind entry and the target of the ACI.

To match when the user attribute on the bind DN entry corresponds directly to the attribute on the target entry, replace *attr* with the user attribute type, and *value* with the attribute value. DS

servers perform an internal search to get the attributes of the bind entry. Therefore, this ACI subject does not work with operational attributes.

To match when the target entry is identified by an LDAP URL, and the bind DN is in the subtree of the DN of the LDAP URL, use `ldap-url#LDAPURL`.

To match when the bind DN corresponds to a member of the group identified by the *attr* value on the target entry, use `attr#GROUPDN`.

To match when the bind DN corresponds to the *attr* value on the target entry, use `attr#USERDN`.

The optional inheritance specification, `parent[child-level].`, lets you specify how many levels below the target entry inherit the ACI. The *child-level* is a number from 0 to 9, with 0 indicating the target entry only. Separate multiple *child-level* digits with commas (`,`).

`userdn [!]= "ldap-url++[|| ldap-url++ …]"`

To match the bind DN, replace *ldap-url*++ with either a valid LDAP URL, such as `ldap:///uid=bjensen,ou=People,dc=example,dc=com`, or `ldap:///dc=example,dc=com??sub?(uid=bjensen)`, or a special LDAP URL-like keyword from the following list:

`ldap:///all`

Match authenticated users.

`ldap:///anyone`

Match anonymous and authenticated users.

`ldap:///parent`

Match when the bind DN is a parent of the ACI target.

`ldap:///self`

Match when the bind DN entry corresponds to ACI target.

How ACI is Evaluated

Understanding how DS servers evaluate the `aci` values is critical when implementing an access control policy.

The rules the server follows are simple:

1. To determine whether an operation is allowed or denied, DS servers look in the directory for the target of the operation. A server collects any ACI values from that entry, and then walks up the directory tree to the suffix, collecting all ACI values en route. Global ACI values are then collected.

2. It then separates the ACI values into two lists; one list contains all the ACI values that match the target and deny the required access, and the other list contains all the ACI values that match the target and allow the required access.
3. If the deny list contains any ACI values after this procedure, access is immediately denied.
4. If the deny list is empty, then the allow list is processed. If the allow list contains any ACI values, access is allowed.
5. If both lists are empty, access is denied.

Note

Some operations require multiple permissions and involve multiple targets. Evaluation will therefore take place multiple times. For example, a search operation requires the `search` permission for each attribute in the search filter. If all those are allowed, the `read` permission is used to decide what attributes and values can be returned.

ACI Required For LDAP Operations

The minimal access control information required for specific LDAP operations is described here:

Add

The ACI must allow the `add` permission to entries in the target. This implicitly allows the attributes and values to be set.

Use `targattrfilters` to explicitly deny access to any values if required.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to add an entry is:

```
aci: (version 3.0;acl "Add entry"; allow (add)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

Bind

Because this is used to establish the user's identity and derived authorizations, ACI is irrelevant for this operation and is not checked.

To prevent authentication, disable the account instead. For details, see "Managing Accounts Manually".

Compare

The ACI must allow the `compare` permission to the attribute in the target entry.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to compare values against the `sn` attribute is:

```
aci: (targetattr = "sn")(version 3.0;acl "Compare surname"; allow (compare)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

Delete

The ACI must allow the `delete` permission to the target entry. This implicitly allows the attributes and values in the target to be deleted.

Use `targattrfilters` to explicitly deny access to the values if required.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to delete an entry is:

```
aci: (version 3.0;acl "Delete entry"; allow (delete)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

Modify

The ACI must allow the `write` permission to attributes in the target entries. This implicitly allows all values in the target attribute to be modified.

Use `targattrfilters` to explicitly deny access to specific values if required.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to modify the `description` attribute in an entry is:

```
aci: (targetattr = "description")(version 3.0; acl "Modify description"; allow (write)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

ModifyDN

If the entry is being moved to a `newSuperior`, the `export` permission must be allowed on the target, and the `import` permission must be allowed on the `newSuperior` entry.

The ACI must allow `write` permission to the attributes in the old RDN and the new RDN. All values of the old RDN and new RDN can be written implicitly; use `targattrfilters` to explicitly deny access to values used if required.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to rename entries named with the `uid` attribute to new locations:

```
aci: (targetattr = "uid")(version 3.0;acl "Rename uid= entries";
 allow (write, import, export)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

Search

ACI is required to process the search filter, and to determine what attributes and values may be returned in the results. The `search` permission is used to allow particular attributes in the search filter. The `read` permission is used to allow particular attributes to be returned.

If `read` permission is allowed to any attribute, the server automatically allows the `objectClass` attribute to also be read.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to search for `uid` attributes, and also to read that attribute in matching entries is:

```
aci: (targetattr = "uid")(version 3.0;acl "Search and read uid"; allow (search, read)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

Use Control or Extended Operation

The ACI must allow the `read` permission to the `targetcontrol` or `extop` OIDs.

For example, the ACI required to allow `uid=bjensen,ou=People,dc=example,dc=com` to use the Persistent Search request control with OID `2.16.840.1.113730.3.4.3` is:

```
aci: (targetcontrol = "2.16.840.1.113730.3.4.3")
 (version 3.0;acl "Request Persistent Search"; allow (read)
 (userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
```

Configuring ACIs

ACIs are defined in the data on `aci` attributes. They can be imported in LDIF and modified over LDAP. In order to make changes to ACIs, however, users first need the `modify-acl` privilege. By default, only the directory superuser has the `modify-acl` privilege.

Global ACIs on `cn=Access Control Handler,cn=config` can be set using the **dsconfig** command. Global ACIs have attribute type `ds-cfg-global-aci`.

Modifying and removing global ACIs can have deleterious effects. Generally the impact depends on your deployment requirements. Modifications to global ACIs fall into the following categories:

- Modification or removal is permitted.

  You must test client applications when deleting the specified ACI.

- Modification or removal may affect applications.

  You must test client applications when modifying or deleting the specified ACI.

- Modification or removal may affect applications, but is not recommended.

  You must test client applications when modifying or deleting the specified ACI.

- Do not modify or delete.

For details, see "Default Global ACIs".

Default Global ACIs

Name	Description	ACI Definition
Anonymous control access	Anonymous and authenticated users can use the LDAP controls that are specified by OID. Modification or removal may affect applications.	`(targetcontrol="2.16.840.1.113730.3.4.2 || 2.16.840.1.113730.3.4.17 || 2.16.840.1.113730.3.4.19 || 1.3.6.1.4.1.4203.1.10.2 || 1.3.6.1.4.1.42.2.27.8.5.1 || 2.16.840.1.113730.3.4.16 || 1.2.840.113556.1.4.1413 || 1.3.6.1.4.1.36733.2.1.5.1") (version 3.0; acl "Anonymous control access"; allow(read) userdn="ldap:///anyone";)`
Anonymous extended operation access	Anonymous and authenticated users can request the LDAP extended operations that are specified by OID. Modification or removal may affect applications.	`(extop="1.3.6.1.4.1.26027.1.6.1 || 1.3.6.1.4.1.26027.1.6.3 || 1.3.6.1.4.1.4203.1.11.1 || 1.3.6.1.4.1.1466.20037 || 1.3.6.1.4.1.4203.1.11.3") (version 3.0; acl "Anonymous extended operation access"; allow(read) userdn="ldap:///anyone";)`
Anonymous read access	Anonymous and authenticated users can read the user data attributes that are specified by their names. Modification or removal is permitted.	`(targetattr!="userPassword||authPassword||debugsearchindex||changes||changeNumber||changeType||changeTime||targetDN||newRDN||newSuperior||deleteOldRDN")(version 3.0; acl "Anonymous read access"; allow (read, search,compare) userdn="ldap:///anyone";)`
Authenticated users control access	Authenticated users can use the LDAP controls that are specified by OID. Modification or removal may affect applications.	`(targetcontrol="1.3.6.1.1.12 || 1.3.6.1.1.13.1 || 1.3.6.1.1.13.2 || 1.2.840.113556.1.4.319 || 1.2.826.0.1.3344810.2.3 || 2.16.840.1.113730.3.4.18 || 2.16.840.1.113730.3.4.9 || 1.2.840.113556.1.4.473 || 1.3.6.1.4.1.42.2.27.9.5.9") (version 3.0; acl "Authenticated users control access"; allow(read) userdn="ldap:///all";)`
Self entry modification	Authenticated users can modify the specified attributes on their own entries. Modification or removal is permitted.	`(targetattr="audio||authPassword||description||displayName||givenName||homePhone||homePostalAddress||initials||jpegPhoto||labeledURI||mobile||pager||postalAddress||postalCode||preferredLanguage||telephoneNumber||userPassword")(version 3.0; acl "Self`

Name	Description	ACI Definition
		`entry modification"; allow (write) userdn="ldap:///self";)`
Self entry read	Authenticated users can read the password values on their own entries. By default, the server applies a one-way hash algorithm to the password value before writing it to the entry, so it is computationally difficult to recover the cleartext version of the password from the stored value. Modification or removal is permitted.	`(targetattr="userPassword||authPassword") (version 3.0; acl "Self entry read"; allow (read,search,compare) userdn="ldap:///self";)`
User-Visible Operational Attributes	Anonymous and authenticated users can read attributes that identify entries and that contain information about modifications to entries. Modification or removal may affect applications.	`(targetattr="createTimestamp|| creatorsName||modifiersName|| modifyTimestamp||entryDN|| entryUUID||subschemaSubentry|| etag||governingStructureRule|| structuralObjectClass||hasSubordinates|| numSubordinates||isMemberOf||alive|| healthy")(version 3.0; acl "User-Visible Operational Attributes"; allow (read, search,compare) userdn="ldap:///anyone";)`
User-Visible Root DSE Operational Attributes	Anonymous and authenticated users can read attributes that describe what the server supports. Modification or removal may affect applications.	`(target="ldap:///")(targetscope="base") (targetattr="objectClass|| namingContexts|| supportedAuthPasswordSchemes|| supportedControl||supportedExtension|| supportedFeatures||supportedLDAPVersion|| supportedSASLMechanisms|| supportedTLSCiphers|| supportedTLSProtocols||vendorName|| vendorVersion")(version 3.0; acl "User-Visible Root DSE Operational Attributes"; allow (read,search,compare) userdn="ldap:///anyone";)`
User-Visible Schema Operational Attributes	Anonymous and authenticated users can read LDAP schema definitions. Modification or removal may affect applications.	`(target="ldap:///cn=schema") (targetscope="base") (targetattr="objectClass|| attributeTypes||dITContentRules|| dITStructureRules||ldapSyntaxes|| matchingRules||matchingRuleUse|| nameForms||objectClasses")(version 3. 0; acl "User-Visible Schema Operational Attributes"; allow (read,search,compare) userdn="ldap:///anyone";)`

Users with write access to add ACIs and with the `modify-acl` privilege can use the **ldapmodify** command to change ACIs located in user data.

This section focuses on ACI examples, rather than demonstrating how to update the directory for each example. To update ACIs, use the **ldapmodify** command.

For hints on updating directory entries with the **ldapmodify** command, see "Modifying Entry Attributes" in the *Developer's Guide*. Keep in mind that the name of the ACI attribute is `aci` as shown in the examples that follow.

This section includes the following examples:

- "ACI: Anonymous Reads and Searches"
- "ACI: Disable Anonymous Access"
- "ACI: Full Access for Administrators"
- "ACI: Change Your Password"
- "ACI: Manage Your Group Membership"
- "ACI: Manage Self-Service Groups"
- "ACI: Permit Cleartext Access Over Loopback Only"
- "ACI: Manage ACI Values"

ACI: Anonymous Reads and Searches

This ACI makes all user attributes world-readable except password attributes:

```
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr != "authPassword || userPassword")
 (version 3.0;acl "Anonymous read-search access";
 allow (read, search, compare)(userdn = "ldap:///anyone");)
```

ACI: Disable Anonymous Access

By default, DS servers deny access unless an ACI explicitly allows access. (This does not apply to the default directory superuser, `cn=Directory Manager`, who has the `bypass-acl` privilege.) DS servers allow anonymous users to read public data and request certain controls and extended operations.

These default capabilities are defined in global ACIs, which you can read by using the following command:

```
$ dsconfig \
 get-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --property global-aci \
 --trustAll
```

You can disable anonymous access either by editing relevant `global-aci` properties, or by using the global server configuration property, `unauthenticated-requests-policy`. Editing relevant `global-aci` properties lets you take a fine-grained approach to limit anonymous access. Setting `unauthenticated-requests-policy:allow-discovery` causes the DS server to reject all requests from clients who are not authenticated except bind requests and StartTLS requests and base object searches on the root DSE.

To take a fine-grained approach, use the **dsconfig** command to edit `global-aci` properties. One of the most expedient ways to do this is to use the command interactively on one DS server, capturing the output to a script with the `--commandFilePath` *`script`* option, and then editing the script for use on other servers. With this approach, you can allow anonymous read access to the root DSE and to directory schemas so that clients do not have to authenticate to discover server capabilities, and also allow anonymous users access to request certain controls and extended operations:

```
#
# Edit Access Control Handler global-aci attributes, replacing
# userdn="ldap:///anyone" (anonymous) with userdn="ldap:///all" (authenticated)
# in "Anonymous read access" and "User-Visible Operational Attributes" ACIs.
#
# To make this change, you remove existing values, and add edited values:
#
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --remove global-aci:\(targetattr!=\"userPassword\|\|authPassword\|\|debugsearchindex\|\|changes\|\|
changeNumber\|\|changeType\|\|changeTime\|\|targetDN\|\|newRDN\|\|newSuperior\|\|deleteOldRDN\"\)\(version
\ 3.0\;\ acl\ \"Anonymous\ read\ access\"\;\ allow\ \(read,search,compare\)\ userdn=\"ldap:///anyone\"\;\)
 \
 --remove global-aci:\(targetattr=\"createTimestamp\|\|creatorsName\|\|modifiersName\|\|modifyTimestamp\|
\|entryDN\|\|entryUUID\|\|subschemaSubentry\|\|etag\|\|governingStructureRule\|\|structuralObjectClass\|
\|hasSubordinates\|\|numSubordinates\|\|isMemberOf\|\|alive\|\|healthy\"\)\(version\ 3.0\;\ acl\ \"User-
Visible\ Operational\ Attributes\"\;\ allow\ \(read,search,compare\)\ userdn=\"ldap:///anyone\"\;\) \
 --add global-aci:\(targetattr!=\"userPassword\|\|authPassword\|\|debugsearchindex\|\|changes\|\|
changeNumber\|\|changeType\|\|changeTime\|\|targetDN\|\|newRDN\|\|newSuperior\|\|deleteOldRDN\"\)\(version
\ 3.0\;\ acl\ \"Anonymous\ read\ access\"\;\ allow\ \(read,search,compare\)\ userdn=\"ldap:///all\"\;\) \
 --add global-aci:\(targetattr=\"createTimestamp\|\|creatorsName\|\|modifiersName\|\|modifyTimestamp\|\|
entryDN\|\|entryUUID\|\|subschemaSubentry\|\|etag\|\|governingStructureRule\|\|structuralObjectClass\|\|
hasSubordinates\|\|numSubordinates\|\|isMemberOf\|\|alive\|\|healthy\"\)\(version\ 3.0\;\ acl\ \"User-
Visible\ Operational\ Attributes\"\;\ allow\ \(read,search,compare\)\ userdn=\"ldap:///all\"\;\) \
 --trustAll \
 --no-prompt
```

Make sure that you also set appropriate ACIs on any data that you import. The following commands prevent anonymous reads of Example.com data:

```
$ cat remove-anon.ldif
dn: dc=example,dc=com
changetype: modify
delete: aci
aci: (target ="ldap:///dc=example,dc=com")(targetattr != "userPassword")(version 3.0;acl "Anonymous read-
search access";allow (read, search, compare)(userdn = "ldap:///anyone");)
-
add: aci
aci: (target ="ldap:///dc=example,dc=com")(targetattr != "userPassword")(version 3.0;acl "Anonymous read-
search access";allow (read, search, compare)(userdn = "ldap:///all");)

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

At this point, clients must authenticate to view search results:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)"
$ ldapsearch  \
 --port 1389 \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" cn uid
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
uid: bjensen
```

To reject anonymous access except bind and StartTLS requests and base object searches on the root DSE, set `unauthenticated-requests-policy:allow-discovery`:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set unauthenticated-requests-policy:allow-discovery \
 --trustAll \
 --no-prompt
```

Once you set the property, anonymous clients trying to search, for example, get an `Unwilling to Perform` response from the DS server:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)"
# The LDAP search request failed: 53 (Unwilling to Perform)
# Additional Information:  Rejecting the requested operation because the connection has not been
 authenticated
```

In both cases, notice that the changes apply to a single DS server configuration, and so are never replicated to other servers. You must apply the changes separately to each replicated server.

ACI: Full Access for Administrators

The following ACI gives a group of Directory Administrators full access. Some administrative operations will also require privileges not shown in this example:

```
aci: (target="ldap:///dc=example,dc=com")
 (targetattr = "* || +")
 (version 3.0;acl "Admins can run amok";
  allow(all, proxy, import, export)
  groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)
```

`targetattr = "* || +"` permits access to all user attributes and all operational attributes. `allow(all, proxy, import, export)` permits all user operations, proxy authorization, and modify DN operations.

ACI: Change Your Password

By default, the right to change one's own password is set in a global ACI. This ACI reproduces the same effect:

```
aci: (target ="ldap:///ou=People,dc=example,dc=com")
 (targetattr = "authPassword || userPassword")
 (version 3.0;acl "Allow users to change pass words";
   allow (write)(userdn = "ldap:///self");)
```

ACI: Manage Your Group Membership

For some static groups, such as carpoolers and social club members, you might choose to let users manage their own memberships. The following ACI lets members of self-service groups manage their own membership:

```
aci: (target ="ldap:///ou=Self Service,ou=Groups,dc=example,dc=com")
 (targetattr = "member")
 (version 3.0;acl "Self registration"; allow(selfwrite)
 (userdn = "ldap:///uid=*,ou=People,dc=example,dc=com");)
```

ACI: Manage Self-Service Groups

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

```
aci: (target ="ldap:///ou=Self Service,ou=Groups,dc=example,dc=com")
 (targattrfilters="add=objectClass:(objectClass=groupOfNames)")
 (version 3.0; acl "All can create self service groups";
   allow (add)(userdn= "ldap:///uid=*,ou=People,dc=example,dc=com");)
aci: (target ="ldap:///ou=Self Service,ou=Groups,dc=example,dc=com")
 (version 3.0; acl "Owner can delete self service groups";
   allow (delete)(userattr= "owner#USERDN");)
```

ACI: Permit Cleartext Access Over Loopback Only

This ACI uses IP address and Security Strength Factor subjects:

```
aci: (target = "ldap:///dc=example,dc=com")
 (targetattr = "*")
 (version 3.0;acl "Use loopback only for LDAP in the clear";
   deny (all)(ip != "127.0.0.1" and ssf <= "1");)
```

When you use TLS but have not configured a cipher, `ssf` is one. Packets are checksummed for integrity checking, but all content is sent in cleartext.

ACI: Manage ACI Values

In order to update ACIs in directory data, a user must have both the `modify-acl` privilege and access to modify the `aci` operational attribute.

This ACI lets Directory Administrators manage `aci` values:

```
aci: (target = "ldap:///dc=example,dc=com")
 (targetattr = "aci") (version 3.0;acl "Modify ACIs"; allow (all)
 (groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)
```

About Global Access Control Policies

Global access control policies are similar to ACIs, but are implemented as entries in the server configuration, rather than attributes on entries. They are managed using the **dsconfig** command.

Unlike ACIs, policies can only allow access, not deny it. This constraint makes them easier to read and to change in isolation. Policies are, however, applied in addition to ACIs and privileges. It is still possible that an ACI could deny something allowed by a policy.

By default, no access is allowed until permitted by a policy or other access control. By default, a policy matches all entries, all types of connection, and all users. You set the properties of the policy to restrict its scope of application. Policies can have the settings to allow the following:

- Requests for specified LDAP controls and extended operations
- Access to specific attributes, with support for wildcards, @objectclass notation, and exceptions to simplify settings
- Read access (for read, search, and compare operations)
- Write access (for add, delete, modify, and modify DN operations)
- Making authentication required or not before requesting an operation
- Requests targeting a particular scope, with wildcards to simplify settings
- Requests originating or not from specific client addresses or domains
- Requests using a specified protocol
- Requests using a specified port
- Requests using a minimum security strength factor
- Requests from a user whose DN does or does not match a DN pattern

For details, see "Global Access Control Policy" in the *Configuration Reference*.

Configuring Global Access Control Policies

You manage global access control policies with the **dsconfig** command. As the policies are part of the server configuration, they are not replicated, and must be configured on each server.

This section includes the following examples:

- "Example Policy: Rejecting Unauthenticated Requests"
- "Example Policy: Require Secure Connections"
- "Example Policy: Allow Anonymous Requests From Specific Network"

Example Policy: Rejecting Unauthenticated Requests

The following example replaces the default policies that allow anonymous access and authenticated access with a single policy for authenticated access. This example then adds a policy to allow

anonymous access to use the StartTLS and Get Symmetric Key extended operations. It does not change the default policy that allows anonymous access to read root DSE operational attributes, as that information should remain publicly readable in most deployments:

```
$ dsconfig \
 delete-global-access-control-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Anonymous access all entries" \
 --trustAll \
 --no-prompt
$ dsconfig \
 delete-global-access-control-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Authenticated access all entries" \
 --trustAll \
 --no-prompt
$ dsconfig \
 create-global-access-control-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Authenticated access all entries" \
 --set authentication-required:true \
 --set request-target-dn-not-equal-to:"**,cn=changelog" \
 --set permission:read \
 --set allowed-attribute:"*" \
 --set allowed-attribute:createTimestamp \
 --set allowed-attribute:creatorsName \
 --set allowed-attribute:entryDN \
 --set allowed-attribute:entryUUID \
 --set allowed-attribute:etag \
 --set allowed-attribute:governingStructureRule \
 --set allowed-attribute:hasSubordinates \
 --set allowed-attribute:isMemberOf \
 --set allowed-attribute:modifiersName \
 --set allowed-attribute:modifyTimestamp \
 --set allowed-attribute:numSubordinates \
 --set allowed-attribute:structuralObjectClass \
 --set allowed-attribute:subschemaSubentry \
 --set allowed-attribute-exception:authPassword \
 --set allowed-attribute-exception:userPassword \
 --set allowed-attribute-exception:debugSearchIndex \
 --set allowed-attribute-exception:@changeLogEntry \
 --set allowed-control:Assertion \
 --set allowed-control:AuthorizationIdentity \
 --set allowed-control:Csn \
 --set allowed-control:ManageDsaIt \
 --set allowed-control:MatchedValues \
 --set allowed-control:Noop \
 --set allowed-control:PasswordPolicy \
 --set allowed-control:PermissiveModify \
```

```
 --set allowed-control:PostRead \
 --set allowed-control:PreRead \
 --set allowed-control:ProxiedAuthV2 \
 --set allowed-control:RealAttributesOnly \
 --set allowed-control:ServerSideSort \
 --set allowed-control:SimplePagedResults \
 --set allowed-control:TransactionId \
 --set allowed-control:VirtualAttributesOnly \
 --set allowed-control:Vlv \
 --set allowed-extended-operation:GetSymmetricKey \
 --set allowed-extended-operation:PasswordModify \
 --set allowed-extended-operation:PasswordPolicyState \
 --set allowed-extended-operation:StartTls \
 --set allowed-extended-operation:WhoAmI \
 --trustAll \
 --no-prompt
$ dsconfig \
 create-global-access-control-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Anonymous extended operation access" \
 --set authentication-required:false \
 --set allowed-extended-operation:GetSymmetricKey \
 --set allowed-extended-operation:StartTls \
 --trustAll \
 --no-prompt
```

Example Policy: Require Secure Connections

The following example creates a policy with a minimum security strength factor of 128, effectively allowing only secure connections for requests targeting data in `dc=example,dc=com`. A security strength factor defines the key strength for DIGEST-MD5, GSSAPI, SSL, and TLS:

```
$ dsconfig \
 create-global-access-control-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Require secure connections for example.com data" \
 --set request-target-dn-equal-to:"**,dc=example,dc=com" \
 --set request-target-dn-equal-to:dc=example,dc=com \
 --set connection-minimum-ssf:128 \
 --trustAll \
 --no-prompt
```

Example Policy: Allow Anonymous Requests From Specific Network

The following example updates policies that allow anonymous requests so they are scoped to apply to clients in the `example.com` domain:

```
$ dsconfig \
 set-global-access-control-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Anonymous extended operation access" \
 --set connection-client-address-equal-to:.example.com \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-global-access-control-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Root DSE access" \
 --set connection-client-address-equal-to:.example.com \
 --trustAll \
 --no-prompt
```

With the `connection-client-address-not-equal-to` property, it is also possible to reject requests from a particular host, domain, address, or address mask.

For additional details, see "Global Access Control Policy" in the *Configuration Reference*.

Viewing Effective Rights

As the number of ACIs increases, it can be difficult to understand by inspection what rights a user actually has to a given entry. The following example shows how the Get Effective Rights control can help.

By default, only users such as Directory Manager who can bypass ACIs can use the Get Effective Rights control (OID `1.3.6.1.4.1.42.2.27.9.5.2`), and related operational attributes, `aclRights` and `aclRightsInfo`. For this example, explicitly grant access to My App using global ACIs:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:\(targetcontrol=\"1.3.6.1.4.1.42.2.27.9.5.2\"\)\ \(version\ 3.0\;acl\ \"Allow\ My\ App\
 to\ get\ effective\ rights\"\;\ allow\(read\)\ userdn=\"ldap:///cn=My\ App,ou=Apps,dc=example,dc=com\"\;
\) \
 --add global-aci:\(targetattr=\"aclRights\|\|aclRightsInfo\"\)\(version\ 3.0\;\ acl\ \"Allow\ My\ App
\ to\ read\ effective\ rights\ attributes\"\;\ allow\ \(read,search,compare\)\ userdn=\"ldap:///cn=My\
 App,ou=Apps,dc=example,dc=com\"\;\) \
 --trustAll \
 --no-prompt
```

In this example, Babs Jensen is the owner of a small group of people who are willing to carpool:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=bjensen,ou=people,dc=example,dc=com" \
 --bindPassword hifalutin \
 --baseDN "ou=Self Service,ou=Groups,dc=example,dc=com" \
 "(cn=Carpoolers)"
dn: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
objectClass: top
objectClass: groupOfNames
description: People who are willing to carpool
cn: Carpoolers
owner: uid=bjensen,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
```

When My App performs the same search with the get effective rights control, and requests the `aclRights` attribute, it sees the rights that it has on the entry:

```
$ ldapsearch \
 --control effectiverights \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --baseDN "ou=Self Service,ou=Groups,dc=example,dc=com" \
 "(cn=Carpoolers)" \
 aclRights
dn: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
aclRights;entryLevel: add:1,delete:1,read:1,write:1,proxy:1
```

When you request the `aclRightsInfo` attribute, the server responds with information about the ACIs applied:

```
$ ldapsearch \
 --control effectiverights \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --baseDN "ou=Self Service,ou=Groups,dc=example,dc=com" \
 "(cn=Carpoolers)" \
 aclRights \
 aclRightsInfo
dn: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
aclRightsInfo;logs;entryLevel;add: acl_summary(main): access allowed(add) on ent
 ry/attr(cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com, NULL) to (cn
 =My App,ou=Apps,dc=example,dc=com) (not proxied) ( reason: evaluated allow , de
 ciding_aci: Allow apps proxied auth)
aclRightsInfo;logs;entryLevel;delete: acl_summary(main): access allowed(delete)
 on entry/attr(cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com, NULL)
 to (cn=My App,ou=Apps,dc=example,dc=com) (not proxied) ( reason: evaluated allo
 w , deciding_aci: Allow apps proxied auth)
aclRightsInfo;logs;entryLevel;read: acl_summary(main): access allowed(read) on e
 ntry/attr(cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com, objectClas
```

```
 s) to (cn=My App,ou=Apps,dc=example,dc=com) (not proxied) ( reason: evaluated a
 llow , deciding_aci: Anonymous read-search access)
aclRightsInfo;logs;entryLevel;write: acl_summary(main): access allowed(write) on
  entry/attr(cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com, NULL) to
  (cn=My App,ou=Apps,dc=example,dc=com) (not proxied) ( reason: evaluated allow
 , deciding_aci: Allow apps proxied auth)
aclRightsInfo;logs;entryLevel;proxy: acl_summary(main): access allowed(proxy) on
  entry/attr(cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com, NULL) to
  (cn=My App,ou=Apps,dc=example,dc=com) (not proxied) ( reason: evaluated allow
 , deciding_aci: Allow apps proxied auth)
aclRights;entryLevel: add:1,delete:1,read:1,write:1,proxy:1
```

You can also request the effective rights for another user by using the `--getEffectiveRightsAuthzid` option (short form: `-g`). This option takes the authorization identity of the user as an argument. The following example shows My App checking Babs's rights to the same entry:

```
$ ldapsearch \
 --getEffectiveRightsAuthzid "dn:uid=bjensen,ou=People,dc=example,dc=com" \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --baseDN "ou=Self Service,ou=groups,dc=example,dc=com" \
 "(cn=Carpoolers)" \
 aclRights
dn: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
aclRights;entryLevel: add:0,delete:1,read:1,write:0,proxy:0
```

The following example checks anonymous user rights to the same entry. Notice that the authorization identity for an anonymous user is expressed as `dn:`:

```
$ ldapsearch \
 --getEffectiveRightsAuthzid "dn:" \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --baseDN "ou=Self Service,ou=groups,dc=example,dc=com" \
 "(cn=Carpoolers)" \
 aclRights
dn: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
aclRights;entryLevel: add:0,delete:0,read:1,write:0,proxy:0
```

To check access to an attribute that might not yet exist on the entry, use the `--getEffectiveRightsAttribute` option (short form: `-e`). This option takes a comma-separated attribute list as an argument. The following example checks Andy Hall's access to the member attribute for the Carpooler's group entry:

```
$ ldapsearch \
 --getEffectiveRightsAuthzid "dn:uid=ahall,ou=People,dc=example,dc=com" \
 --getEffectiveRightsAttribute member \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --baseDN "ou=Self Service,ou=groups,dc=example,dc=com" \
 "cn=Carpoolers" \
 aclRights
dn: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
aclRights;attributeLevel;member:
 search:1,read:1,compare:1,write:0,selfwrite_add:1,selfwrite_delete:1,proxy:0
aclRights;entryLevel: add:0,delete:0,read:1,write:0,proxy:0
```

Chapter 7

Indexing Attribute Values

This chapter covers DS indexing features used to speed up searches, and to limit the impact of searches on directory server resources. In this chapter you will learn to:

- Define indexes and explain why they are useful
- Determine what to index and what types of indexes to use
- Configure, build, and rebuild indexes
- Check that indexes are valid

About Indexes

A basic, standard directory feature is the ability to respond quickly to searches.

An LDAP search specifies the following information that directly affects how long the directory might take to respond:

- The base DN for the search.

  The more specific the base DN, the less information to check during the search. For example, a request with base DN `dc=example,dc=com` potentially involves checking many more entries than a request with base DN `uid=bjensen,ou=people,dc=example,dc=com`.

- The scope of the search.

  A subtree or one-level scope targets many entries, whereas a base search is limited to one entry.

- The search filter to match.

  A search filter, such as `(cn=Babs Jensen)`, asserts that an attribute on the entry to search for, in this case `cn`, corresponds to some value. In this case, the attribute must have a value that equals `Babs Jensen`, ignoring case sensitivity.

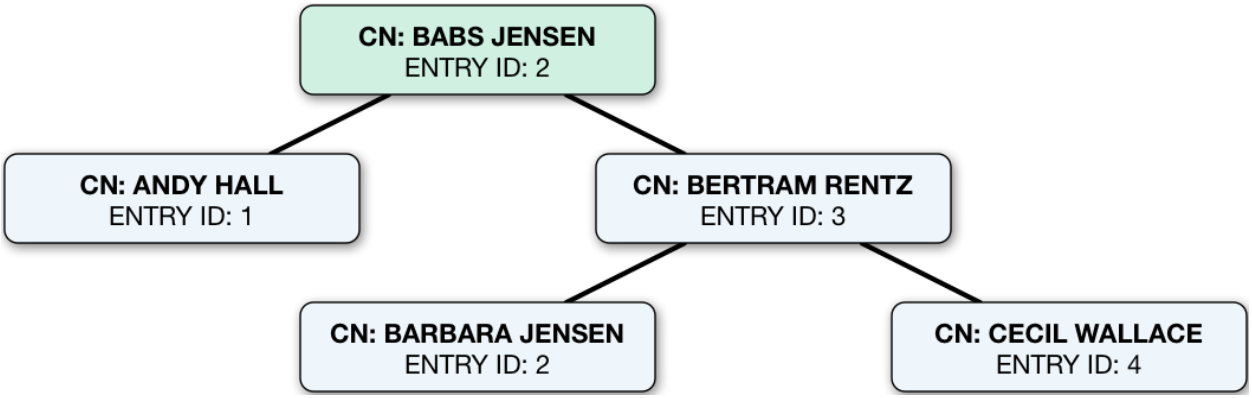
  It would generally be a waste of resources to have the directory server check all entries to see whether they have a CN of Babs Jensen. Instead, directory servers maintain indexes to expedite checking whether a search filter matches.

LDAP directory servers like DS directory servers even go so far as to disallow searches that cannot be handled expediently using indexes. Maintaining appropriate indexes is a key aspect of directory administration.

The role of an index is to answer the question, "Which entries have an attribute with this corresponding value?" Each index is therefore specific to an attribute. Each index is also specific to the comparison implied in the search filter. For example, a directory server maintains distinct indexes for exact (equality) matching and for substring matching. The types of indexes are explained in "Index Types and Their Functions". Furthermore, indexes are configured in specific directory backends.

An index is implemented as a tree of key-value pairs. The key is a form of the value to match, such as `babs jensen`. The value is a list of IDs for entries that match the key. "An Equality Index" shows an equality (case ignore exact match) index with five keys from a total of four entries. If the data set were large, there could be more than one entry ID per key.

An Equality Index



This is how DS directory servers use indexes. When the search filter is `(cn=Babs Jensen)`, the directory server retrieves the IDs for entries with a CN matching `Babs Jensen` from the equality index of the CN attribute. (For a complex filter, the directory server might optimize the search by changing the order in which it uses the indexes.) A successful result is zero or more entry IDs. These are the candidate result entries.

For each candidate, the DS directory server retrieves the entry by ID from a special system index called `id2entry`, which, as its name suggests, returns an entry for an entry ID. If there is a match, and the client application has the right to access to the data, the directory server returns the search result. It continues this process until no candidates are left.

If there are no indexes that correspond to a search request, then the DS directory server must potentially check for a match against every entry in the scope of the search. Evaluating every entry for a match is referred to as an *unindexed* search. An unindexed search is an expensive operation, particularly for large directories. For this reason, a directory server refuses unindexed searches unless the user making the request has specific permission to make such requests. Permission to

perform an unindexed search is granted with the `unindexed-search` privilege. This privilege is reserved for the directory root user by default, and should not be granted lightly.

If the number of user data entries is smaller than the default resource limits, you can still perform what appear to be unindexed searches, meaning searches with filters for which no index appears to exist. That is because the `dn2id` index returns all user data entries without hitting a resource limit that would make the search unindexed.

Use cases that may call for unindexed searches include the following:

- An application must periodically retrieve a very large amount of directory data all at once through an LDAP search.

  For example, an application performs an LDAP search to retrieve everything in the directory once a week as part of a batch job that runs during off hours.

  Make sure the application has no resource limits. For details, see "*Setting Resource Limits*".

- A directory data administrator occasionally browses directory data through a graphical UI without initially knowing what they are looking for or how to narrow the search.

  For this case, DS directory servers can sort unindexed search results as long as they are paged.

  This capability has the following limitations:

  - The simple paged results control must specify a *page size* that is less than or equal to the `index-entry-limit` (default: 4000).
  - For each page, the server reads the entire backend database, retaining *page size* number of sorted entries.

  Alternatively, DS directory servers can use an appropriately configured VLV index to sort results for an unindexed search. For details, see "VLV Index for Paged Server-Side Sort".

What To Index

DS directory server search performance depends on indexes as described in "About Indexes".

DS directory servers maintain generally useful indexes for data imported into the default backend. When you create a new backend, the directory server only maintains the necessary system indexes unless you configure additional indexes. For details, see "Default Indexes".

The default settings are fine for evaluating DS software, and they work well with sample data. The default settings might not, however, fit your directory data and the searches performed on your directory service.

You can manage indexes using the command-line tools demonstrated in "Configuring and Rebuilding Indexes".

Determining Which Indexes Are Needed

Index maintenance has its costs. Every time an indexed attribute is updated, the DS directory server must update each affected index to reflect the change, which is wasteful if the index is hardly used. Indexes, especially substring indexes, can take up more memory and disk space than the corresponding data.

Aim to maintain only those indexes that speed up appropriate searches, and that allow the DS directory server to operate properly. The latter indexes include non-configurable internal indexes, and generally are handled by the directory server without intervention. The former, indexes for appropriate searches, require thought and investigation. Whether a search is appropriate depends on the circumstances.

Begin by reviewing the attributes of your directory data. Which attributes would you expect to see in a search filter? If an attribute is going to show up frequently in reasonable search filters, then it ought to be indexed.

Compare your guesses with what you see actually happening in the directory. One way of doing this is to review the access log for search results that are marked with additional items including `unindexed`:

```
$ grep unindexed /path/to/opendj/logs/ldap-access.audit.json
{...,"request":{"protocol":"LDAP","operation":"SEARCH",..."detail":"You do not have sufficient privileges
 to perform an unindexed search","additionalItems":"unindexed"...}
{...,"request":{"protocol":"LDAP","operation":"SEARCH",..."detail":"You do not have sufficient privileges
 to perform an unindexed search","additionalItems":"unindexed"...}
{...,"request":{"protocol":"LDAP","operation":"SEARCH",..."detail":"You do not have sufficient privileges
 to perform an unindexed search","additionalItems":"unindexed"...}
```

Read the full messages in the access log on your server, as they also specify the search filter and scope. Understand the search that led to each unindexed search. If the filter is appropriate and frequently used, add an index to facilitate the search. You can either consume the access logs to determine how often a search filter is used, or monitor what is happening in the directory by using the index analysis feature.

DS servers provide this feature to collect information about filters in search requests. You can activate the index analysis mechanism using the **dsconfig set-backend-prop** command:

```
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --set index-filter-analyzer-enabled:true \
 --no-prompt \
 --trustAll
```

The command causes the server to analyze filters used, and to keep the results in memory, so that you can read them through the `cn=monitor` interface:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN ds-cfg-backend-id=userRoot,cn=Backends,cn=monitor \
 "(objectclass=*)" \
 filter-use
dn: ds-cfg-backend-id=userRoot,cn=Backends,cn=monitor
filter-use: (sn=Ishee) hits:1 maxmatches:8 message:
filter-use: (mail=*.com) hits:1 maxmatches:-1 message:The filter value exceeded the index entry limit for
 the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 index
filter-use: (uid=user.86182) hits:1 maxmatches:1 message:
filter-use: (objectclass=person) hits:2 maxmatches:-1 message:The filter value exceeded the index entry
 limit for the /dc=com,dc=example/objectClass.objectIdentifierMatch index
filter-use: (employeenumber=86182) hits:1 maxmatches:-1 message:equality index type is disabled for the
 employeeNumber attribute
filter-use: (mail=*@example.com) hits:1 maxmatches:-1 message:The filter value exceeded the index entry
 limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter value exceeded
 the index entry limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter
 value exceeded the index entry limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6
 indexThe filter value exceeded the index entry limit for the /dc=com,dc=example/
mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter value exceeded the index entry limit for the /
dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter value exceeded the index entry limit
 for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter value exceeded the index
 entry limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter value exceeded
 the index entry limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 indexThe filter
 value exceeded the index entry limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6
 indexThe filter value exceeded the index entry limit for the /dc=com,dc=example/
mail.caseIgnoreIA5SubstringsMatch:6 index
```

The `filter-use` values are the filter, the `hits` (number of times the filter was used), the `maxmatches` (number of matches found), and an optional message.

The output can include filters for internal use, such as `(aci=*)`. In the example above, you see filters for client application searches.

One appropriate search filter that led to an unindexed search, `(employeenumber=86182)`, had no matches because, "equality index type is disabled for the employeeNumber attribute." Some client application is trying to find specific users by employee number, but no index exists for that purpose. If this appears regularly as a frequent search, add an employee number index as described in "Configuring a Standard Index".

One inappropriate search filter that led to an unindexed search, `(mail=*.com)`, had no matches because, "The filter value exceeded the index entry limit for the /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6 index." It appears that some client application is trying to list all entries with an email address ending in `.com`. There are so many such entries that although an index exists for the `mail` attribute, the server has given up maintaining the list of entries with email addresses ending in `.com`. In a large directory, there might be many thousands of matching entries. If you take action to allow this expensive search, the requests could consume a large share of directory resources, or even cause a denial of service to other requests.

To avoid impacting server performance, turn off index analysis after you collect the information you need. You turn off index analysis with the **dsconfig set-backend-prop** command:

```
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --set index-filter-analyzer-enabled:false \
 --no-prompt \
 --trustAll
```

Directory users might complain to you that their searches are refused because they are unindexed. Ask for the result code, additional information, and search filter. DS directory servers respond to LDAP client applications that attempt unindexed searches with a result code of 50 and additional information about the unindexed search. The following example attempts, anonymously, to get the entries for all users whose email address ends in `.com`:

```
$ ldapsearch \
 --port 1389 \
 --baseDN ou=people,dc=example,dc=com \
 "(&(mail=*.com)(objectclass=person))"
# The LDAP search request failed: 50 (Insufficient Access Rights)
# Additional Information:  You do not have sufficient privileges to perform an unindexed search
```

Rather than adjusting settings to permit the search, try to understand why the user wants to perform an unindexed search.

Perhaps they are unintentionally requesting an unindexed search. If so, you can help them find a less expensive search, by using an approach that limits the number of candidate result entries. For example, if a GUI application lets a user browse a group of entries, the application could use a browsing index to retrieve a block of entries for each screen, rather than retrieving all the entries at once.

Perhaps they do have a legitimate reason to get the full list of all entries in one operation, such as regularly rebuilding some database that depends on the directory. If so, their application can perform the search as a user who has the `unindexed-search` privilege. To assign the `unindexed-search` privilege, see "Configuring Privileges".

In addition to searches a server performs in response to client search requests, a server also performs internal searches. Internal searches let the server retrieve data needed to respond to a request, or maintain internal state information. In some cases internal searches can become unindexed. When this happens, the server logs a warning similar to the following:

```
The server is performing an unindexed internal search request
with base DN '%s', scope '%s', and filter '%s'.
Unindexed internal searches are usually unexpected and could impact performance.
Please verify that that backend's indexes are configured correctly
for these search parameters.
```

When you see a message like this in the server log, take these actions:

- Figure out which indexes are missing as described in "Clarifying Which Indexes Are Used by a Search".

  If any indexes are missing, add them as described in "Configuring and Rebuilding Indexes".

- Check the integrity of the indexes used as described in "Verifying Indexes".

- If the relevant indexes exist and you have verified that they are sound, it could be that an index entry limit setting is too low.

  This can happen, for example, in directory servers having more than 4000 groups in a single backend. For details, see "Understanding Index Entry Limits".

- If you have made the changes described in the steps above, but you continue to see the warning messages and problem persists, contact technical support.

Clarifying Which Indexes Are Used by a Search

Sometimes it is not obvious by inspection how a directory server handles a given search request internally. The directory root user can inspect how the DS directory server resolves the search request by performing the same search with the `debugsearchindex` attribute.

A default global access control setting prevents users from reading the `debugsearchindex` attribute. To allow an administrator to read the attribute, add a global access control setting as in the following example for a directory server using ACIs:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetattr=\"debugsearchindex\")(version 3.0; acl \"Debug search indexes\"; \
  allow (read,search,compare) userdn=\"ldap:///uid=user.0,ou=people,dc=example,dc=com\";)" \
 --trustAll \
 --no-prompt
```

Note

The format of `debugsearchindex` values has interface stability: Internal. (See "ForgeRock Product Stability Labels" in the *Release Notes*.)

The values are intended to be read by human beings, not scripts. If you do write scripts that interpret `debugsearchindex` values, be aware that they are not stable. Be prepared to adapt your scripts for every upgrade or patch.

The `debugsearchindex` attribute value indicates how the server would process the search. The server use its indexes to prepare a set of candidate entries. It iterates through the set to compare candidates

with the search filter, returning entries that match. The following example demonstrates this feature for a subtree search with a complex filter:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=user.0,ou=people,dc=example,dc=com \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(&(objectclass=person)(givenName=aa*))" \
 debugsearchindex | sed -n -e "s/^debugsearchindex: //p"
{
  "filter": {
    "intersection": [
      {
        "index": "givenName.caseIgnoreSubstringsMatch:6",
        "range": "[aa,ab[",
        "candidates": 171,
        "retained": 171
      },
      {
        "index": "givenName.caseIgnoreMatch",
        "range": "[aa,ab[",
        "candidates": 50,
        "retained": 50
      },
      {
        "filter": "(objectclass=person)",
        "union": [
          {
            "index": "objectClass.objectIdentifierMatch",
            "exact": "person",
            "candidates": "[LIMIT-EXCEEDED]"
          }
        ],
        "candidates": "[LIMIT-EXCEEDED]",
        "retained": 50
      }
    ],
    "candidates": 50
  },
  "scope": {
    "type": "sub",
    "candidates": "[NOT-INDEXED]",
    "retained": 50
  },
  "final": 50
}
```

The filter in the example matches person entries whose given name starts with `aa`. The search scope is not explicitly specified, so the scope defaults to the subtree including the base DN.

Notice that the `debugsearchindex` value has the following top-level fields:

- (Optional) `"vlv"` describes how the server uses VLV indexes.

  The VLV field is not applicable for this example, and so is not present.

- `"filter"` describes how the server uses the search filter to narrow the set of candidates.
- `"scope"` describes how the server uses the search scope.
- `"final"` indicates the final number of candidates in the set.

In the output, notice that the server uses the equality and substring indexes to find candidate entries whose given name starts with `aa`. If the filter indicated given names *containing* `aa`, as in `givenName=*aa*`, the server would rely only on the substring index.

Notice that the output for the `(objectclass=person)` portion of the filter shows `"candidates": "[LIMIT-EXCEEDED]"`. In this case, there are so many entries matching the value specified that the index is not useful for narrowing the set of candidates. The scope is also not useful for narrowing the set of candidates. Ultimately, however, the `givenName` indexes help the server to narrow the set of candidates. The overall search is indexed and the result is 50 matching entries.

The following example shows a subtree search for accounts with initials starting either with `aa` or with `zz`:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --bindDN uid=user.0,ou=people,dc=example,dc=com \
 --bindPassword password \
 "(|(initials=aa*)(initials=zz*))" \
 debugsearchindex | sed -n -e "s/^debugsearchindex: //p"
{
  "filter": {
    "union": [
      {
        "filter": "(initials=aa*)",
        "index": "initials.presence",
        "diagnostic": "not indexed",
        "candidates": "[NOT-INDEXED]"
      }
    ],
    "candidates": "[NOT-INDEXED]"
  },
  "scope": {
    "type": "sub",
    "candidates": "[NOT-INDEXED]",
    "retained": "[NOT-INDEXED]"
  },
  "final": "[NOT-INDEXED]"
}
```

As shown in the output, the search is not indexed. As described in "Configuring and Rebuilding Indexes", you can index the `initials` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --index-name initials \
 --set index-type:equality \
 --trustAll \
 --no-prompt
$ rebuild-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --baseDn dc=example,dc=com \
 --index initials \
 --trustAll
```

After configuring and building the new index, try the same search again:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --bindDN uid=user.0,ou=people,dc=example,dc=com \
 --bindPassword password \
 "(|(initials=aa*)(initials=zz*))" \
 debugsearchindex | sed -n -e "s/^debugsearchindex: //p"
{
  "filter": {
    "union": [
      {
        "filter": "(initials=aa*)",
        "intersection": [
          {
            "index": "initials.caseIgnoreSubstringsMatch:6",
            "range": "[aa,ab[",
            "diagnostic": "not indexed",
            "tryPresenceIndex": {
              "index": "initials.presence",
              "diagnostic": "not indexed"
            },
            "candidates": "[NOT-INDEXED]",
            "retained": "[NOT-INDEXED]"
          },
          {
            "index": "initials.caseIgnoreMatch",
            "range": "[aa,ab[",
            "candidates": 378,
            "retained": 378
          }
        ],
        "candidates": 378
      },
      {
```

```
        "filter": "(initials=zz*)",
        "intersection": [
          {
            "index": "initials.caseIgnoreSubstringsMatch:6",
            "range": "[zz,z{[",
            "diagnostic": "not indexed",
            "tryPresenceIndex": {
              "index": "initials.presence",
              "diagnostic": "not indexed"
            },
            "candidates": "[NOT-INDEXED]",
            "retained": "[NOT-INDEXED]"
          },
          {
            "index": "initials.caseIgnoreMatch",
            "range": "[zz,z{[",
            "candidates": 26,
            "retained": 26
          }
        ],
        "candidates": 26
      }
    ],
    "candidates": 404
  },
  "scope": {
    "type": "sub",
    "candidates": "[NOT-INDEXED]",
    "retained": 404
  },
  "final": 404
}
```


Notice that the server can narrow the list of candidates using the equality index you created. The server would require a substring index instead of an equality index if the filter were not matching initial strings.

If an index already exists, but you suspect it is not working properly, see "Verifying Indexes".

Index Types and Their Functions

DS directory servers support multiple index types, each corresponding to a different type of search. This section describes the index types and what they are used for.

View what is indexed by using the **backendstat list-indexes** command. For details about a particular index, you can use the **backendstat dump-index** command.

Presence Index

A presence index is used to match an attribute that is present on the entry, regardless of the value. The `aci` attribute is indexed for presence by default to allow quick retrieval of entries with ACIs:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(aci=*)" \
 aci
```

Due to its implementation, a presence index takes up less space than other indexes. In a presence index, there is just one key with a list of IDs.

As described in "About Indexes", an index is implemented as a tree of key-value pairs. The following command examines the ACI presence index for a directory server set up in evaluation mode:

```
$ stop-ds --quiet
$ backendstat \
 dump-index \
 --backendId dsEvaluation \
 --baseDn dc=example,dc=com \
 --indexName aci.presence
Key (len 1): PRESENCE
Value (len 4): [COUNT:3] 1 9 11

Total Records: 1
Total / Average Key Size: 1 bytes / 1 bytes
Total / Average Data Size: 4 bytes / 4 bytes
```

In this case, entries with ACI attributes have IDs `1`, `9`, and `11`.

Equality Index

An equality index is used to match values that correspond exactly (though generally without case sensitivity) to the value provided in the search filter. An equality index requires clients to match values without wildcards or misspellings:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" mail
dn: uid=bjensen,ou=People,dc=example,dc=com
mail: bjensen@example.com
```

An equality index has one list of entry IDs for each attribute value. Depending on the backend implementation, the keys in a case-insensitive index might not be strings. For example, a key of `6A656E73656E` could represent `jensen`.

As described in "About Indexes", an index is implemented as a tree of key-value pairs. The following command examines the SN equality index for a directory server set up in evaluation mode:

```
$ stop-ds --quiet
$ backendstat \
 dump-index \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 --indexName sn.caseIgnoreMatch
...
Key (len 6): jensen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
```

In this case, there are nine entries that have an SN of Jensen.

As long as the keys of the equality index are not encrypted, a directory server can reuse an equality index for some other searches, such as ordering and initial substring searches.

Approximate Index

An approximate index is used to match values that "sound like" those provided in the filter. An approximate index on `cn` lets client applications find people even when they misspell names, as in the following example:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn~=Babs Jansen)" cn
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
```

An approximate index squashes attribute values into a normalized form.

As described in "About Indexes", an index is implemented as a tree of key-value pairs. The following command examines an SN approximate index added to a directory server set up in evaluation mode:

```
$ stop-ds --quiet
$ backendstat \
 dump-index \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 --indexName sn.ds-mr-double-metaphone-approx
...
Key (len 4): JNSN
Value (len 11): [COUNT:10] 18 31 32 59 66 79 94 133 134 150
...
```

In this case, there are ten entries that have an SN that sounds like Jensen.

Substring Index

A substring index is used to match values that are specified with wildcards in the filter. Substring indexes can be expensive to maintain, especially for large attribute values:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=Barb*)" cn
dn: uid=bfrancis,ou=People,dc=example,dc=com
cn: Barbara Francis

dn: uid=bhal2,ou=People,dc=example,dc=com
cn: Barbara Hall

dn: uid=bjablons,ou=People,dc=example,dc=com
cn: Barbara Jablonski

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

dn: uid=bmaddox,ou=People,dc=example,dc=com
cn: Barbara Maddox
```

In a substring index, there are enough keys to allow the server to match any substring in the attribute values. Each key is associated with a list of IDs. The default maximum size of a substring key is 6 bytes.

As described in "About Indexes", an index is implemented as a tree of key-value pairs. The following command examines an SN substring index for a directory server set up in evaluation mode:

```
$ stop-ds --quiet
$ backendstat \
dump-index \
 --backendID dsEvaluation \
 --baseDN dc=example,dc=com \
 --indexName sn.caseIgnoreSubstringsMatch:6
...
Key (len 1): e
Value (len 25): [COUNT:22] ...
...
Key (len 2): en
Value (len 15): [COUNT:12] ...
...
Key (len 3): ens
Value (len 3): [COUNT:1] 147
Key (len 5): ensen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
Key (len 6): jensen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
Key (len 1): n
Value (len 35): [COUNT:32] ...
...
Key (len 2): ns
Value (len 3): [COUNT:1] 147
Key (len 4): nsen
```

```
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
Key (len 1): s
Value (len 13): [COUNT:12] 12 26 47 64 95 98 108 131 135 147 149 154
...
Key (len 2): se
Value (len 7): [COUNT:6] 52 58 75 117 123 148
Key (len 3): sen
Value (len 10): [COUNT:9] 18 31 32 66 79 94 133 134 150
...
```

In this case, the SN value Jensen shares substrings with many other entries. Given the size of the lists and number of keys in a substring index, it is much more expensive to maintain than other indexes. This is particularly true for longer attribute values.

Ordering Index

An ordering index is used to match values for a filter that specifies a range. For example, the `ds-sync-hist` attribute, which is for the server's internal use, has an ordering index by default. Searches on that attribute often seek entries with changes more recent than the last time a search was performed.

The following example shows a search that specifies a range on the SN attribute value:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(sn>=winter)" sn
dn: uid=aworrell,ou=People,dc=example,dc=com
sn: Worrell

dn: uid=kwinters,ou=People,dc=example,dc=com
sn: Winters

dn: uid=pworrell,ou=People,dc=example,dc=com
sn: Worrell
```

In this case, the server only requires an ordering index if it cannot reuse the (ordered) equality index instead. For example, if the equality index is encrypted, the ordering index would need to be maintained separately.

Virtual List View (Browsing) Index

A virtual list view (VLV) or browsing index is designed to help the server respond to client applications that need virtual list view results, for example, to browse through a long list in a GUI. They also help the server respond to clients that request server-side sorting of the search results.

VLV indexes correspond to particular searches. Configure your VLV indexes using the command-line.

Extensible Matching Rule Index

In some cases you need an index for a matching rule other than those described above. For example, DS servers support generalized time-based matching so that applications can search for all times later than, or earlier than a specified time.

Configuring and Rebuilding Indexes

You modify index configurations by using the **dsconfig** command. The subcommands to use depend on the backend type, as shown in the examples that follow. The configuration changes then take effect after you rebuild the index according to the new configuration, using the **rebuild-index** command. The **dsconfig --help-database** command lists subcommands for creating, reading, updating, and deleting index configuration.

Tip

Indexes are per directory backend rather than per suffix. To maintain separate indexes for different suffixes on the same directory server, put the suffixes in different backends.

This section includes the following procedures:

- "Configuring a Standard Index"
- "Configuring a Virtual List View Index"
- "Rebuilding Indexes"
- "Understanding Index Entry Limits"

Configuring a Standard Index

You can configure standard indexes on the command-line using the **dsconfig** command. After you finish configuring the index, you must rebuild the index for the changes to take effect.

To prevent indexed values from appearing in cleartext in a backend, you can enable confidentiality by backend index. For details, see "Encrypting Directory Data".

This section includes the following examples:

- "Create a New Index"
- "Add an Approximate Index"
- "Configure an Extensible Match Index"

Create a New Index

The following example creates a new equality index for the `description` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --index-name description \
 --set index-type:equality \
 --trustAll \
 --no-prompt
```

Add an Approximate Index

The following example add an approximate index for the `sn` (surname) attribute:

```
$ dsconfig \
 set-backend-index-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name sn \
 --add index-type:approximate \
 --trustAll \
 --no-prompt
```

Approximate indexes depend on the Double Metaphone matching rule, described in "Configure an Extensible Match Index".

Configure an Extensible Match Index

DS servers support matching rules defined in LDAP RFCs. They also define DS-specific extensible matching rules.

The following are DS-specific extensible matching rules:

Name: `ds-mr-double-metaphone-approx`
OID: `1.3.6.1.4.1.26027.1.4.1`

Double Metaphone Approximate Match described at http://aspell.net/metaphone/. The DS implementation always produces a single value rather than one or possibly two values.

Configure approximate indexes as described in "Add an Approximate Index".

For an example using this matching rule, see "Search: Finding an Approximate Match" in the *Developer's Guide*.

Name: `ds-mr-user-password-exact`
OID: `1.3.6.1.4.1.26027.1.4.2`

User password exact matching rule used to compare encoded bytes of two hashed password values for exact equality.

Name: `ds-mr-user-password-equality`
OID: `1.3.6.1.4.1.26027.1.4.3`

User password matching rule implemented as the user password exact matching rule.

Name: `partialDateAndTimeMatchingRule`
OID: `1.3.6.1.4.1.26027.1.4.7`

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

For an example using this matching rule, see "Search: Listing Active Accounts" in the *Developer's Guide*.

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

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

For an example that configures an index with this matching rule, see "Search: Listing Active Accounts" in the *Developer's Guide*.

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

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

For an example using this matching rule, see "Search: Listing Active Accounts" in the *Developer's Guide*.

The following example configures an extensible matching rule index for "later than" and "earlier than" generalized time matching on a `lastLoginTime` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --set index-type:extensible \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.5 \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.6 \
 --index-name lastLoginTime \
 --trustAll \
 --no-prompt
```

Configuring Indexes for JSON Attributes

DS servers support attribute values that have JSON syntax. The following schema excerpt defines a `json` attribute with case-insensitive matching:

```
attributeTypes: ( json-attribute-oid NAME 'json'
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY caseIgnoreJsonQueryMatch
  SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
```

When you index a JSON attribute defined in this way, the default directory server behavior is to maintain index keys for each JSON field. Large or numerous JSON objects can result in large indexes, which is wasteful. If you know which fields are used in search filters, you can choose to index only those fields.

As described in "Working With JSON", for some JSON objects only a certain field or fields matter when comparing for equality. In these special cases, the server can ignore other fields when checking equality during updates, and you would not maintain indexes for other fields.

How you index a JSON attribute depends on the matching rule in the attribute's schema definition, and on the JSON fields you expect to be used as search keys for the attribute.

This section includes the following elements:

- "To Use an Indexed JSON Attribute"
- "Example: Custom Index Using a JSON Query Matching Rule"
- "Example: Index Using a Custom JSON Equality Matching Rule"

To Use an Indexed JSON Attribute

The examples that follow demonstrate these steps:

1. Using the schema definition and the information in the following table, configure a custom schema provider for the attribute's matching rule, if necessary.

Matching Rule in Schema Definition	Fields in Search Filter	Custom Schema Provider Required?
`caseExactJsonQueryMatch` `caseIgnoreJsonQueryMatch`	Any JSON field	No
`caseExactJsonQueryMatch` `caseIgnoreJsonQueryMatch`	Specific JSON field or fields	Yes See "Example: Custom Index Using a JSON Query Matching Rule"
`caseExactJsonIdMatch` `caseIgnoreJsonIdMatch`	"_id" field only	No See "Example: Custom Index Using a JSON Query Matching Rule"

Matching Rule in Schema Definition	Fields in Search Filter	Custom Schema Provider Required?
Custom JSON equality matching rule	Specific field(s)	Yes See "Example: Index Using a Custom JSON Equality Matching Rule"

Bear in mind that a custom schema provider applies to all attributes using this matching rule.

2. Add the schema definition for the JSON attribute.
3. Configure the index for the JSON attribute.
4. Add the JSON attribute values in the directory data.

Example: Custom Index Using a JSON Query Matching Rule

This example illustrates the steps in "To Use an Indexed JSON Attribute".

The following command configures a custom, case-insensitive JSON query matching rule. The implementation is like that of the `caseIgnoreJsonQueryMatch` rule, but it only maintains keys for the `access_token` and `refresh_token` fields:

```
$ dsconfig \
 create-schema-provider \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Custom JSON Query Matching Rule" \
 --type json-query-equality-matching-rule \
 --set enabled:true \
 --set case-sensitive-strings:false \
 --set ignore-white-space:true \
 --set matching-rule-name:caseIgnoreJsonQueryMatch \
 --set matching-rule-oid:1.3.6.1.4.1.36733.2.1.4.1 \
 --set indexed-field:access_token \
 --set indexed-field:refresh_token \
 --trustAll \
 --no-prompt
```

The following commands add schemas for a `json` attribute that uses the matching rule:

```
$ cat json-schema.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( json-attribute-oid NAME 'json'
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY caseIgnoreJsonQueryMatch
  SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( json-object-class-oid NAME 'jsonObject' SUP top
  AUXILIARY MAY ( json ) X-ORIGIN 'DS Documentation Examples' )

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

The following command configures an index using the custom matching rule implementation:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --index-name json \
 --set index-type:equality \
 --trustAll \
 --no-prompt
```

For an example of how a client application could use this index, see "Search: Using JSON Query Filters" in the *Developer's Guide*.

Example: Index Using a Custom JSON Equality Matching Rule

This example illustrates the steps in "To Use an Indexed JSON Attribute".

The following command configures a custom, case-insensitive JSON equality matching rule, `caseIgnoreJsonTokenIdMatch`. The implementation is the same as that of the `caseIgnoreJsonIdMatch` rule, except that it indexes the `id` rather than the `_id` field:

```
$ dsconfig \
 create-schema-provider \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Custom JSON Token ID Matching Rule" \
 --type json-equality-matching-rule \
 --set enabled:true \
 --set case-sensitive-strings:false \
 --set ignore-white-space:true \
 --set matching-rule-name:caseIgnoreJsonTokenIDMatch \
 --set matching-rule-oid:1.3.6.1.4.1.36733.2.1.4.4.1 \
 --set json-keys:id \
 --trustAll \
 --no-prompt
```

Notice that this example defines a matching rule with OID `1.3.6.1.4.1.36733.2.1.4.4.1`. In production deployments, use a numeric OID allocated for your own organization.

The following commands add schemas for a `jsonToken` attribute, where the unique identifier is in the "id" field of the JSON object:

```
$ cat json-token-schema.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( jsonToken-attribute-oid NAME 'jsonToken'
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY caseIgnoreJsonTokenIDMatch
  SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( json-token-object-class-oid NAME 'JsonTokenObject' SUP top
  AUXILIARY MAY ( jsonToken ) X-ORIGIN 'DS Documentation Examples' )

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

The following command configures an index using the custom matching rule implementation:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --index-name jsonToken \
 --set index-type:equality \
 --trustAll \
 --no-prompt
```

For an example of how a client application could use this index, see "Search: Using a JSON Assertion" in the *Developer's Guide*.

Configuring a Virtual List View Index

You can create the VLV index configuration by using the **dsconfig** command.

The following example shows how to create the VLV index. This example applies when it is clear that GUI users will browse user accounts, sorting on surname then given name:

```
$ dsconfig \
 create-backend-vlv-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name people-by-last-name \
 --set base-dn:ou=People,dc=example,dc=com \
 --set filter:"(|(givenName=*)(sn=*))" \
 --set scope:single-level \
 --set sort-order:"+sn +givenName" \
 --trustAll \
 --no-prompt
```

Note

When referring to a VLV index after creation, you must add `vlv.` as a prefix. In other words, if you named the VLV index `people-by-last-name`, you refer to it as `vlv.people-by-last-name` when rebuilding indexes, changing index properties such as the index entry limit, or verifying indexes.

VLV Index for Paged Server-Side Sort

A special VLV index can enable the server to sort results for a search that is technically *unindexed*. For example, this feature facilitates paging through an entire directory database in a UI, where the user does not necessarily filter the data before knowing what is available.

Because the search is technically unindexed, the user performing the search must have the `unindexed-search` privilege.

The VLV index must have the following characteristics:

- Its filter must be "always true," `(&)`.
- Its scope must cover the search scope of the requests.
- Its base DN must match or be a parent of the base DN of the search requests.
- Its sort order must match the sort keys of the requests in the order they occur in the requests, starting with the first sort key used in the request. The VLV index sort order can include additional keys not present in a request.

  In other words, if the sort order of the VLV index is `+l +roomNumber +sn +cn`, then it works with requests having the following sort orders:

  `+l +roomNumber +sn +cn`
  `+l +roomNumber +sn`
  `+l +roomNumber`
  `+l`

The following example commands demonstrate creating and using a VLV index to sort paged results by locality, room number, surname, and then full name. The `l` and `roomNumber` attributes are not indexed by default. This example makes use of the **rebuild-index** command described below. It gives Kirsten Vaughan the `unindexed-search` privilege. It also allows Kirsten Vaughan to use the server-side sort control, which is not allowed by default:

```
$ dsconfig \
 create-backend-vlv-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name by-last-name \
 --set base-dn:ou=People,dc=example,dc=com \
 --set filter:"(&)" \
 --set scope:subordinate-subtree \
 --set sort-order:"+l +roomNumber +sn +cn" \
 --trustAll \
 --no-prompt
$ rebuild-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --baseDn dc=example,dc=com \
 --index vlv.by-last-name \
 --trustAll
$ cat allow-unindexed-search.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
```

```
add: ds-privilege-name
ds-privilege-name: unindexed-search

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 allow-unindexed-search.ldif
$ cat allow-server-side-sort.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol = "2.16.840.1.113730.3.4.3")
 (version 3.0;acl "Allow Server-Side Sort for Kirsten Vaughan";
 allow (read)(userdn = "ldap:///uid=kvaughan,ou=People,dc=example,dc=com");)

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 allow-server-side-sort.ldif
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDn uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDn dc=example,dc=com \
 --sortOrder +l,+roomNumber,+sn,+cn \
 --simplePageSize 5 \
 "(&)" \
 cn l roomNumber
dn: uid=cnewport,ou=People,dc=example,dc=com
cn: Christoph Newport
l: Bristol
roomNumber: 0056

dn: uid=awhite,ou=People,dc=example,dc=com
cn: Alan White
l: Bristol
roomNumber: 0142

dn: uid=rdaugherty,ou=People,dc=example,dc=com
cn: Robert Daugherty
l: Bristol
roomNumber: 0194

dn: uid=mlott,ou=People,dc=example,dc=com
cn: Mike Lott
l: Bristol
roomNumber: 0498

dn: uid=brentz,ou=People,dc=example,dc=com
cn: Bertram Rentz
l: Bristol
roomNumber: 0617

Press RETURN to continue ^C
```

Rebuilding Indexes

After you change an index configuration, or when you find that an index is corrupt, you can rebuild the index. When you rebuild indexes, you specify the base DN of the data to index, and either the list of indexes to rebuild or `--rebuildAll`. You can rebuild indexes while the server is offline, or while the server is online. If you rebuild the index while the server is online, then you must schedule the rebuild process as a task.

Important

When you rebuild an index with the server online, the server takes the index's backend database offline while it rebuilds the index.

The data that the backend database serves is unavailable to client operations during this time.

This section includes the following examples:

- "Rebuild Index"
- "Rebuild Degraded Indexes"
- "Clear New, Unused, Degraded Indexes"

Rebuild Index

The following example rebuilds the `cn` index immediately with the server online:

```
$ rebuild-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 --index cn \
 --trustAll
```

Rebuild Degraded Indexes

The following example rebuilds degraded indexes immediately with the server online:

```
$ rebuild-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 --rebuildDegraded \
 --trustAll
```

Clear New, Unused, Degraded Indexes

When you add a new attribute as described in "Updating Directory Schema", and then create indexes for the new attribute, the new indexes appear as degraded, even though the attribute has not yet been used, and so indexes are sure to be empty, rather than degraded.

In this special case, you can safely use the **rebuild-index --clearDegradedState** command to avoid having to scan the entire directory backend before rebuilding the new, unused index. In this example, an index has just been created for `newUnusedAttribute`.

Before using the **rebuild-index** command, test the index status to make sure that the index has not yet been used: by using the **backendstat** command, described in "*backendstat — gather OpenDJ backend debugging information*" in the *Reference*.

DS servers must be stopped before you use the **backendstat** command:

```
$ stop-ds --quiet
```

The third column of the output is the `Valid` column, which is `false` before the rebuild:

```
$ backendstat \
 show-index-status \
 --backendID userRoot \
 --baseDN dc=example,dc=com \
 | grep newUnusedAttribute
newUnusedAttribute.presence                    ... false ...
newUnusedAttribute.caseIgnoreMatch             ... false ...
newUnusedAttribute.caseIgnoreSubstringsMatch:6 ... false ...
```

Update the index information to fix the value of the unused index:

```
$ rebuild-index \
 --offline \
 --baseDN dc=example,dc=com \
 --clearDegradedState \
 --index newUnusedAttribute
```

Check that the `Index Valid` column for the index status is now set to `true`:

```
$ backendstat \
 show-index-status \
 --backendID userRoot \
 --baseDN dc=example,dc=com \
 | grep newUnusedAttribute
newUnusedAttribute.presence                    ... true ...
newUnusedAttribute.caseIgnoreMatch             ... true ...
newUnusedAttribute.caseIgnoreSubstringsMatch:6 ... true ...
```

Start the server:

```
$ start-ds --quiet
```

If the newly indexed attribute has already been used, rebuild the index instead of clearing the degraded state.

Understanding Index Entry Limits

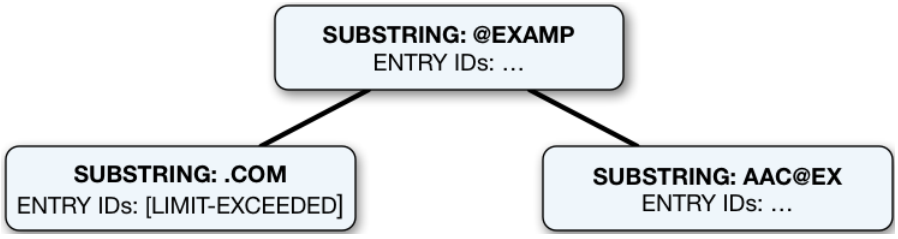
As described in "About Indexes", an index is implemented as a tree of key-value pairs. The key is what the search is trying to match. The value is a list of entry IDs.

As the number of entries in the directory grows, the list of entry IDs for some keys can become very large. For example, every entry in the directory has the value `top` for the `objectClass` attribute. If the directory maintains a substring index for `mail`, the number of entries ending in `.com` could be huge.

A directory server therefore defines an *index entry limit*. When the number of entry IDs for a key exceeds the limit, the server stops maintaining a list of IDs for that key. The limit effectively makes a search using that key unindexed. Searches using other keys in the same index are not affected.

"Index Entry Limit Exceeded For a Single Key" shows a fragment from a substring index for the `mail` attribute. The number of email addresses ending in `.com` has exceeded the index entry limit. For the other substring keys, the entry ID lists are still maintained, but to save space the entry IDs are not shown in the diagram.

Index Entry Limit Exceeded For a Single Key



Ideally, the limit is set at the point where it becomes more expensive to maintain the entry ID list for a key and to perform an indexed search than to perform an unindexed search. In practice, the limit is a trade off, with a default index entry limit value of 4000.

To Check Index Entry Limits

The following steps show how to get information about indexes where the index entry limit is exceeded for some keys. In this case, the directory server holds 100,000 user entries. The settings for this directory server are reasonable.

Use the **backendstat show-index-status** command, described in "*backendstat — gather OpenDJ backend debugging information*" in the *Reference*.

1. Stop DS servers before you use the **backendstat** command:

```
$ stop-ds --quiet
```

2. Non-zero values in the Over Entry Limit column of the output table indicate the number of keys for which the limit has been reached. The keys that are over the limit are then listed below the table:

```
$ backendstat show-index-status --backendID userRoot --baseDN dc=example,dc=com
Index Name                                      ... Valid  ... Over Entry Limit ...
...
mail.caseIgnoreIA5SubstringsMatch:6             ... true   ... 34 ...
...
givenName.caseIgnoreSubstringsMatch:6           ... true   ... 9  ...
telephoneNumber.telephoneNumberSubstringsMatch:6 ... true   ... 10 ...
...
sn.caseIgnoreSubstringsMatch:6                  ... true   ... 14 ...
...
cn.caseIgnoreSubstringsMatch:6                  ... true   ... 14 ...
objectClass.objectIdentifierMatch               ... true   ... 4  ...
...

Index: /dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6
Over index-entry-limit keys: [.com] [0@mail] ...

Index: /dc=com,dc=example/givenName.caseIgnoreSubstringsMatch:6
Over index-entry-limit keys: [a] [e] [i] [ie] [l] [n] [na] [ne] [y]

Index: /dc=com,dc=example/telephoneNumber.telephoneNumberSubstringsMatch:6
Over index-entry-limit keys: [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

Index: /dc=com,dc=example/cn.caseIgnoreSubstringsMatch:6
Over index-entry-limit keys: [a] [an] [e] [er] [i] [k] [l] [n] [o] [on] [r] [s] [t] [y]

Index: /dc=com,dc=example/objectClass.objectIdentifierMatch
Over index-entry-limit keys: [inetorgperson] [organizationalperson] [person] [top]

Index: /dc=com,dc=example/sn.caseIgnoreSubstringsMatch:6
```

```
Over index-entry-limit keys: [a] [an] [e] [er] [i] [k] [l] [n] [o] [on] [r] [s] [t] [y]
```

For example, every user entry has the object classes listed, and every user entry has an email address ending in `.com`, so those values are not specific enough to be used in search filters.

3. Start the server:

```
$ start-ds --quiet
```

Index Entry Limit Changes

In rare cases, the index entry limit might be too low for a certain key. This could manifest itself as a frequent, useful search becoming unindexed, with no reasonable way to narrow the search.

You can change the index entry limit on a per-index basis. Do not do this in production unless you can explain and show why the benefits outweigh the costs.

Important

Changing the index entry limit significantly can result in serious performance degradation. Be prepared to test performance thoroughly before you roll out an index entry limit change in production.

Consider a directory with more than 4000 groups in a backend. When the backend is brought online, a directory server searches for the groups with a search filter of `(|(objectClass=groupOfNames)(objectClass=groupOfEntries)(objectClass=groupOfUniqueNames))`, which is an unindexed search due to the default index entry limit setting. The following example raises the index entry limit for the `objectClass` index to `10000`, and then rebuilds the index for the configuration change to take effect. The steps are the same for any other index:

```
$ dsconfig \
 set-backend-index-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --index-name objectClass \
 --set index-entry-limit:10000 \
 --no-prompt \
 --trustAll
$ stop-ds --quiet
$ rebuild-index \
 --baseDN dc=example,dc=com \
 --index objectClass \
 --offline
$ start-ds --quiet
```

It is also possible, but not recommended, to configure the global `index-entry-limit` for a backend. This changes the default for all indexes in the backend. Use the **dsconfig set-backend-prop** command as shown in the following example:

```
# Not recommended
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --set index-entry-limit:10000 \
 --trustAll \
 --no-prompt
```

Verifying Indexes

You can verify that indexes correspond to current directory data, and that indexes do not contain errors by using the **verify-index** command, described in "*verify-index — check index for consistency or errors*" in the *Reference*.

Verify Index

The following example verifies the `cn` index offline for completeness and for errors:

```
$ stop-ds --quiet
$ verify-index \
 --baseDN dc=example,dc=com \
 --index cn \
 --clean \
 --countErrors
```

The output indicates whether any errors are found in the index.

Default Indexes

When you first install a directory server and import your data from LDIF, the indexes are configured as shown in "Default Indexes".

Default Indexes

Index	Approx.	Equality	Ordering	Presence	Substring	Entry Limit
`aci`	-	-	-	Yes	-	4000

Index	Approx.	Equality	Ordering	Presence	Substring	Entry Limit
cn	-	Yes	-	-	Yes	4000
dn2id	Non-configurable internal index					
ds-sync-conflict	-	Yes	-	-	-	4000
ds-sync-hist	-	-	Yes	-	-	4000
entryUUID	-	Yes	-	-	-	4000
givenName	-	Yes	-	-	Yes	4000
id2children	Non-configurable internal index					
id2subtree	Non-configurable internal index					
mail	-	Yes	-	-	Yes	4000
member	-	Yes	-	-	-	4000
objectClass	-	Yes	-	-	-	4000
sn	-	Yes	-	-	Yes	4000
telephone-Number	-	Yes	-	-	Yes	4000
uid	-	Yes	-	-	-	4000
uniqueMember	-	Yes	-	-	-	4000

When you create a new backend using the **dsconfig** command, DS directory servers create the following indexes automatically:

`aci` presence
`ds-sync-conflict` equality
`ds-sync-hist` ordering
`entryUUID` equality
`objectClass` equality

You can create additional indexes as described in "Configuring and Rebuilding Indexes".

Chapter 8

Managing Data Replication

DS software uses advanced data replication with automated conflict resolution to help ensure your directory services remain available during administrative operations that take an individual server offline, or in the event a server crashes or a network goes down. This chapter explains how to manage DS directory data replication. In this chapter you will learn to:

- Understand how replication operates in order to configure it appropriately
- Configure, initialize, and stop data replication
- Configure standalone directory servers and replication servers, or break a server that plays both roles into two standalone servers
- Configure replication groups, read-only replicas, subtree replication, and fractional replication for complex deployments
- Configure and use change notification to synchronize external applications with changes to directory data
- Recover from situations where a user error has been applied to all replicas
- Resolve replication conflicts

About Replication

Read this section to learn more about how DS replication works.

Replication Defined

Replication is the process of copying updates between DS servers such that all directory servers converge on identical copies of directory data. Replication is designed to let convergence happen over time by default. Letting convergence happen over time means that different replicas can be momentarily out of sync. It also means that if you lose an individual server or even an entire data center, your directory service can keep on running, and then get back in sync when the servers are restarted or the network is repaired.

Replication is specific to DS software. Replication uses a specific protocol that replays update operations quickly, storing enough historical information about updates to resolve most conflicts

automatically. For example, if two client applications separately update a user entry to change the phone number, replication can identify the latest change, and apply it on all replicas without human intervention. To prevent it from growing forever, DS servers purge historical information periodically (default interval: three days). As a directory administrator, you must make sure that the servers do not purge historical information more often than you back up the directory data.

DS server software supports replication over LAN and WAN networks. If you have multiple sites with many servers communicating over the WAN, be sure to read "Standalone Replication Servers" for tips on limiting traffic between sites.

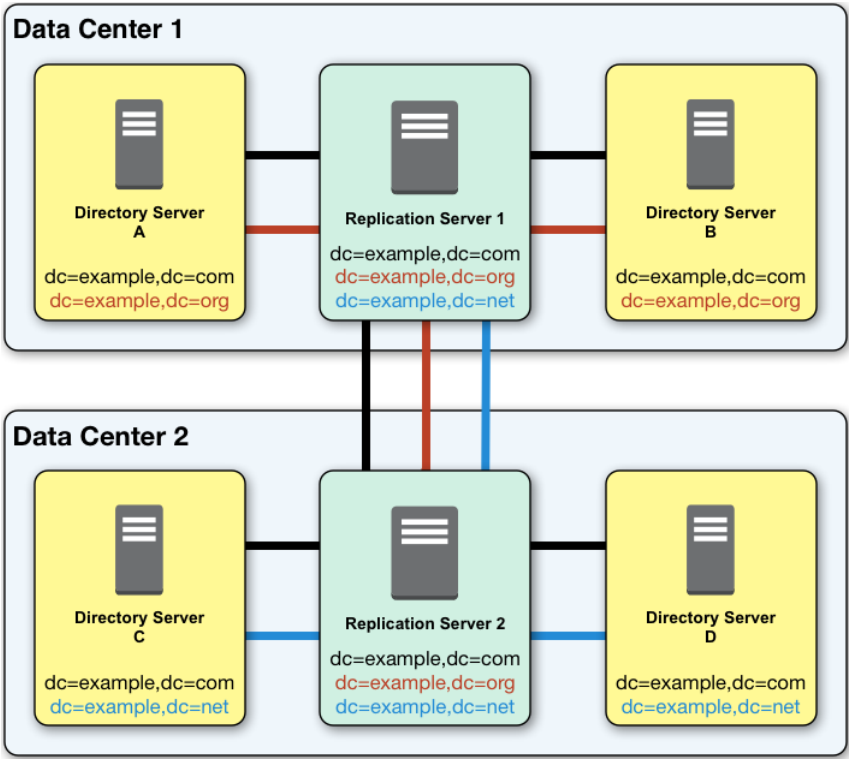
Keep server clocks synchronized for your topology. You can use `ntpd`, for example. Keeping server clocks synchronized helps prevent issues with secure connections and with replication itself. Keeping server clocks synchronized also makes it easier to compare timestamps from multiple servers.

Replication Per Suffix

The primary unit of replication is the suffix, specified by a base DN such as `dc=example,dc=com`. (When you configure partial and fractional replication, however, you can replicate only part of a suffix, or only certain attributes on entries. Also, if you split your suffix across multiple backends, then you need to set up replication separately for each part of suffix in a different backend.) Replication also depends on the directory schema, defined on `cn=schema`, and the `cn=admin data` suffix with administrative identities and certificates for protecting communications. That content gets replicated as well.

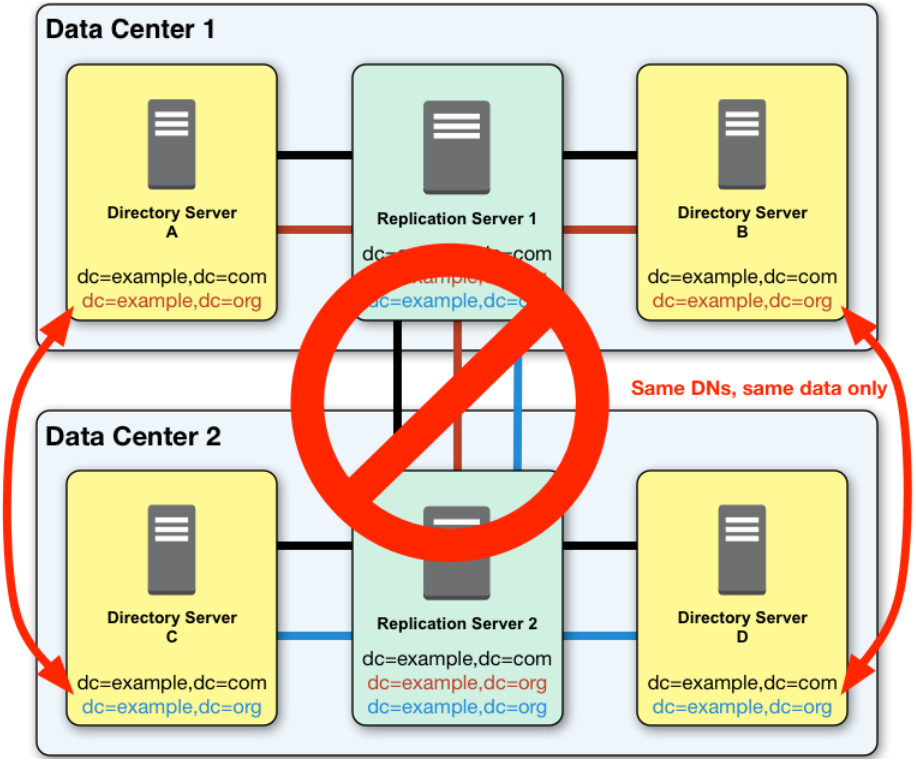
The set of DS servers replicating data for a given suffix is called a *replication topology*. You can have more than one replication topology. For example, one topology could be devoted to `dc=example,dc=com`, and another to `dc=example,dc=org`. DS servers serve more than one suffix, and participate in more than one replication topology as shown in "Correct Replication Topology Configuration".

Correct Replication Topology Configuration



Within a replication topology, the suffixes being replicated are identified to the replication servers by their DNs. All the replication servers are fully connected in a topology. Consequently it is impossible to have multiple separate, independent topologies for data under the same DN within the overall set of servers. This is illustrated in "Incorrect Replication Topology Configuration".

Incorrect Replication Topology Configuration



Replication Network Use and Operations

DS servers listen on dedicated ports for administrative requests and for replication requests. These dedicated ports must remain open to remote requests from configuration tools and from other servers.

DS server configuration tools securely connect to administration ports. Administrative connections are short-lived. When configuring replication or reading replication status, a single **dsreplication** command can connect to multiple servers.

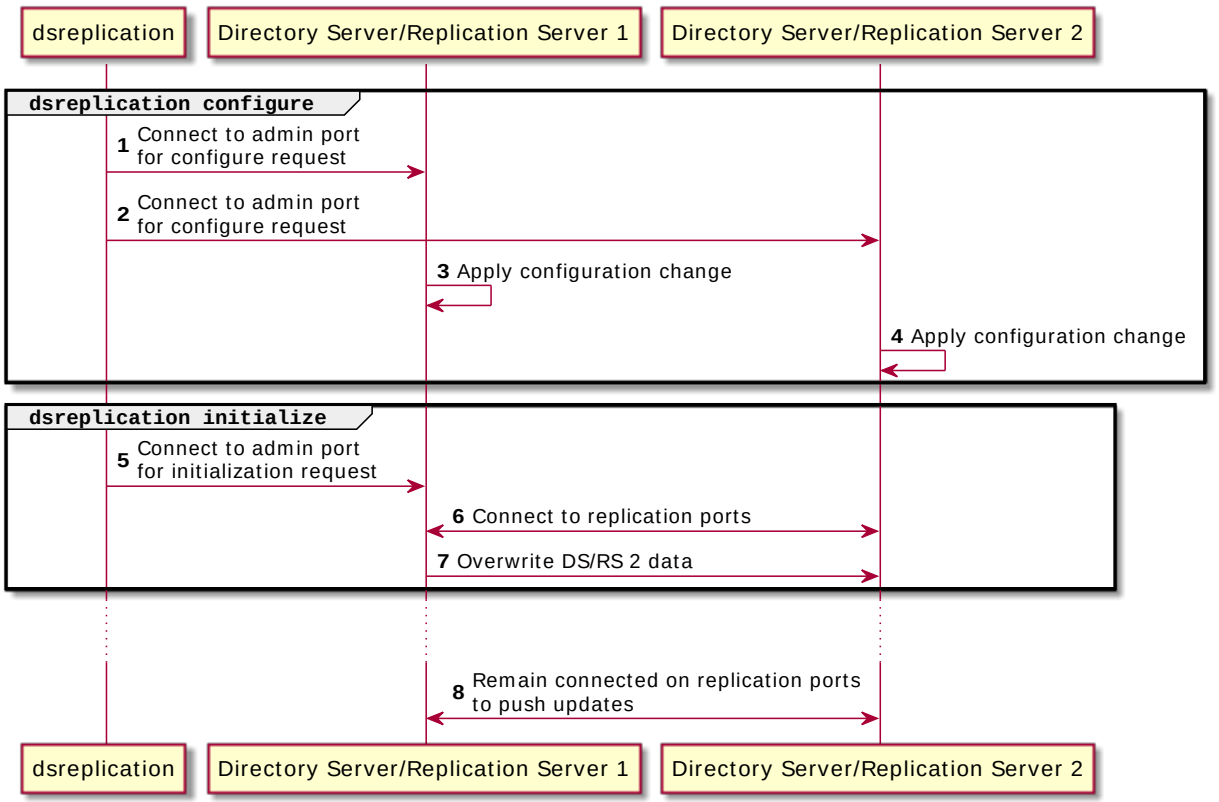
DS directory server replicas connect to DS replication ports for replication requests. A server listening on a replication port is called a *replication server*, whether it is running inside the same process as a directory server, or on a separate host system. Replication connections are long-lived. Each DS replica connects to a replication server upon initialization and then at startup time. The replica keeps the connection open to push and receive updates, although it can connect to another replication server as detailed in "Replication Connection Selection".

The following sequence diagrams show how key operations involve the network. These operations are the following:

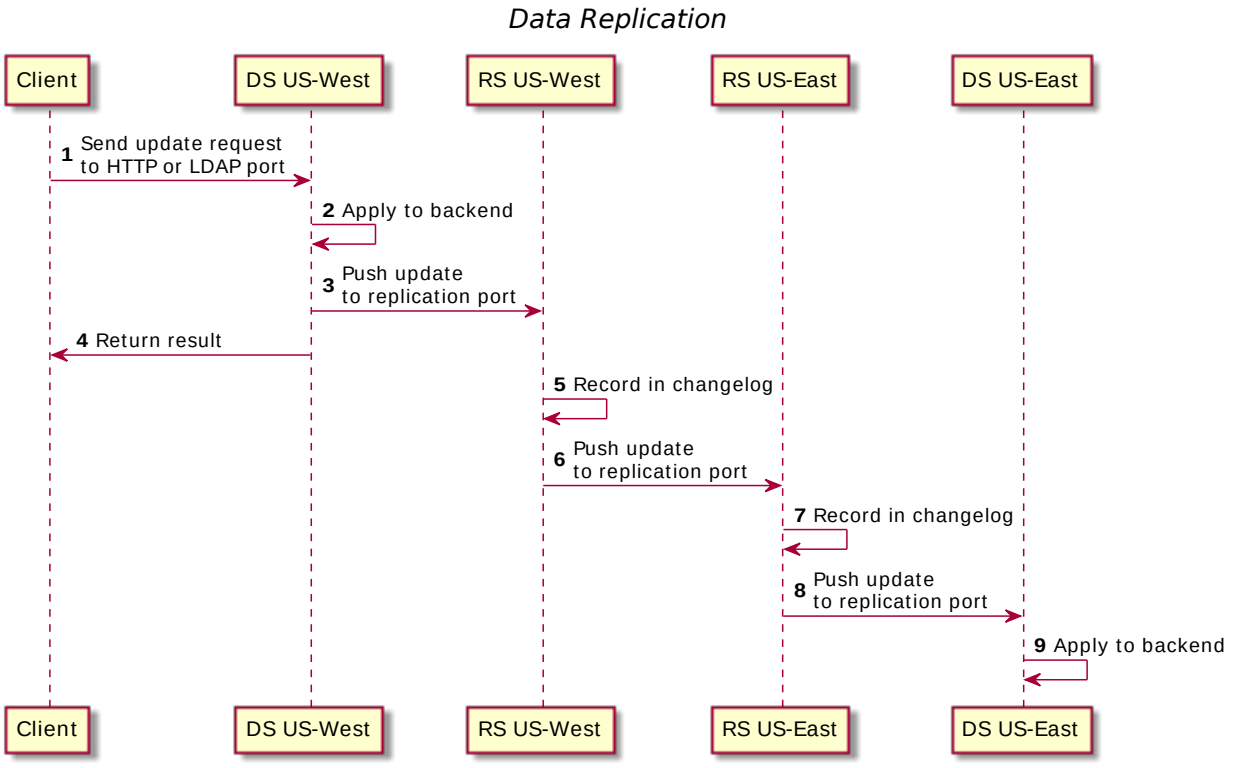
- Setting up replication (configuration, then initialization)
- Data replication (normal operation on update involving no administrative intervention)
- Checking replication status

The commands to configure and to initialize replication use the administrative and replication ports, respectively, as shown in the following sequence diagram:

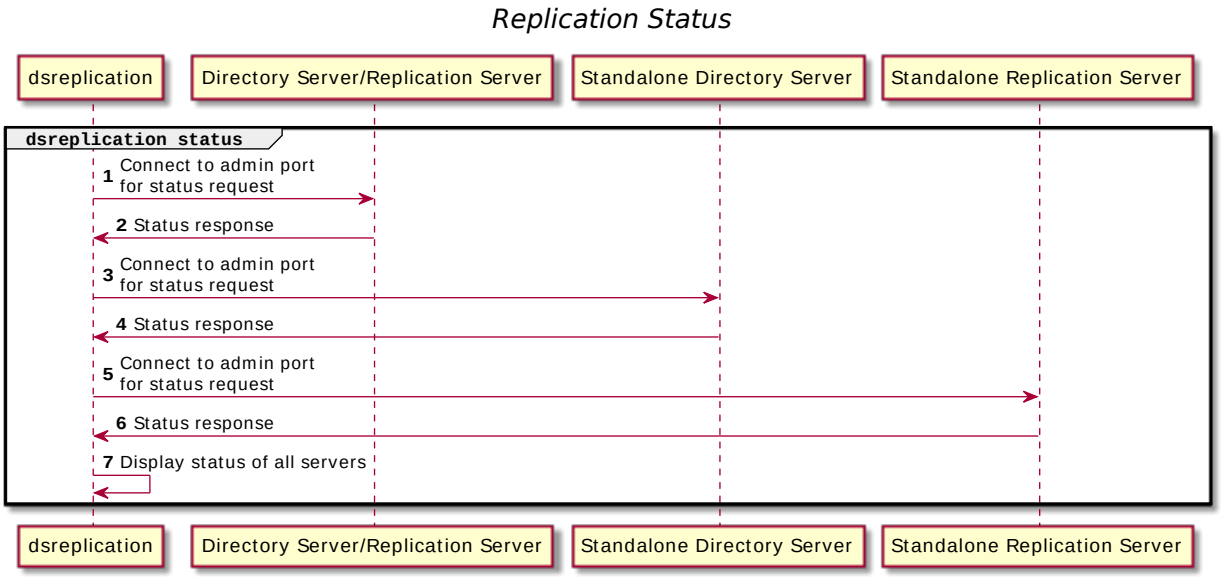
Replication Configuration and Initialization



DS replicas push updates to and receive updates from replication servers over replication port connections. When processing an update, a directory server (DS) replica pushes it to the replication server (RS) it is connected to. The replication server pushes the update to connected directory server replicas and to other replication servers. Notice that directory servers do not connect directly to each other, but through replication servers. This is shown in the following sequence diagram:



The command to monitor replication status uses the administration ports on multiple servers, as shown in the following sequence diagram:



"Limiting System and Administrative Access" in the *Security Guide* provides detailed descriptions of the ports used.

Replication Connection Selection

In order to understand what happens when individual servers stop responding due to a network partition or a crash, know that DS servers can simultaneously provide directory services and replication services. The two services are not the same, even if they can run alongside each other in the same DS server in the same JVM.

Replication relies on the replication service provided by DS replication servers, where DS directory servers publish changes made to their data, and subscribe to changes published by other DS directory servers. The replication service manages replication data only, handling replication traffic with DS directory servers and with other replication servers, receiving, sending, and storing only changes to directory data rather than directory data itself. Once a replication server is connected to a replication topology, it maintains connections to all other replication servers in that topology.

The directory service handles directory data. It responds to requests, and stores directory data and historical information. For each replicated suffix, such as `dc=example,dc=com`, `cn=schema`, and `cn=admin data`, the directory service publishes changes to a replication service, and subscribes to changes from that replication service. (Directory services do not publish changes directly to other directory services.) A directory server also resolves any conflicts that arise when reconciling changes from other directory servers, using the historical information about changes to resolve the conflicts. (Conflict resolution is the responsibility of the directory server rather than the replication server.)

Once a directory server is connected to a replication topology for a particular suffix, it connects to one replication server at a time for that suffix. The replication server provides the directory server with a list of all replication servers for that suffix. Given the list of possible replication servers it can connect to, the directory server can determine which replication server to connect to when starting up, or when the current connection is lost or becomes unresponsive.

For each replicated suffix, a directory server prefers to connect to a replication server:

1. Running in the same JVM as the directory server
2. In the same group as the directory server
3. Having the same initial data for the suffix as the directory server
4. If initial data was the same, a replication server having all the latest changes from the directory server
5. Having the most available capacity relative to other eligible replication servers

   Available capacity depends on how many directory servers in the topology are already connected to a replication server, and what proportion of all directory servers in the topology ought to be connected to the replication server.

   To determine what proportion of the total number of directory servers should be connected to a replication server, a directory server uses replication server weight. When configuring a replication server, you can assign it a weight (default: 1). The weight property takes an integer that indicates capacity to provide replication service relative to other servers. For example, a weight of 2 would indicate a replication server that can handle twice as many connected servers as a replication server with weight 1.

   The proportion of directory servers in a topology that should be connected to a given replication server is equal to (replication server weight)/(sum of replication server weights). In other words, if there are four replication servers in a topology each with default weights, the proportion for each replication server is 1/4.

Consider a situation where seven directory servers are connected to replication servers A, B, C, and D for `dc=example,dc=com` data. Suppose two directory servers each are connected to A, B, and C, one directory server is connected to replication server D. Replication server D is therefore the server with the most available capacity relative to other replication servers in the topology. All other criteria being equal, replication server D is the server to connect to when an eighth directory server joins the topology.

The directory server regularly updates the list of replication servers in case it must reconnect. As available capacity of replication servers for each replication topology can change dynamically, a directory server can potentially reconnect to another replication server to balance the replication load in the topology. For this reason, the server can also end up connected to different replication servers for different suffixes.

Configuring Replication Settings

This section shows how to configure replication with command-line tools, such as the **dsreplication** command, described in "*dsreplication — manage directory data replication*" in the *Reference*.

Configuring Replication

Before configuring replication for a server, optionally set the global server ID. Replication uses this to uniquely identify the server, and therefore the ID must be unique. If you do not set the global server ID, then the configuration process assigns multiple random unique IDs per server. These are harder to remember and to track than an ID that you assign. The ID you assign must be unique across the replication topology. For example, the following command sets the global server ID for a single server to 1:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set server-id:1 \
 --trustAll \
 --no-prompt
```

You can configure a server to replicate with other servers by using the **dsreplication configure** command:

```
$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 opendj.example.com \
 --port1 4444 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --replicationPort1 8989 \
 --host2 replica.example.com \
 --port2 4444 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --replicationPort2 8989 \
 --trustAll \
 --no-prompt
...
Replication has been successfully configured.
...
```

When you configure replication, you update the configuration of both replicas. Do not run multiple **dsreplication configure** commands at the same time.

Important

When you set **dsreplication** hostname options, such as `--hostname`, `--host1`, and `--host2`, make sure to always use either FQDNs or IP addresses. Use whichever are most stable and well-known to servers in your environment. (Never use local-only names or addresses, such as `localhost`, for production systems as they do not uniquely identify the server to other servers.)

Make sure that you use the same hostname value that was provided at installation time to the **setup** command. If you do not know the original hostname used at installation time, read the entry with DN `cn=DIGEST-MD5, cn=SASL Mechanisms,cn=config`. Its `ds-cfg-server-fqdn` attribute holds the original hostname.

Furthermore, always use the same hostname value for the same server each time you invoke the **dsreplication** command. The server uses the values you specify in its configuration. When you use one value to configure replication and another to unconfigure replication, the latter operation fails on remote servers, leaving the system in a broken state.

If you do not know the original hostname values used with the **dsreplication** command, read the entry with DN `cn=all-servers,cn=Server Groups,cn=admin data`. Its `uniqueMember` attribute holds the hostname values.

To enable secure connections for replication use the `--secureReplication1`, and `--secureReplication2` options.

At this point, replication is configured but not active. You must initialize replication in order to replicate data with other servers.

Tip

When scripting the configuration to set up multiple replicas in quick succession, use the same initial replication server each time you run the command. In other words, pass the same `--host1`, `--port1`, `--bindDN1`, `--bindPassword1`, and `--replicationPort1` options for each of the other replicas that you set up in your script.

To replicate with another server, use **dsreplication configure** with the new server as the second server.

To Configure Replication Interactively

You can use the **dsreplication** command interactively.

Tip

When setting up trust between replicas with unrecognized certificates, such as the default self-signed certificates used for replication, the order makes a difference. You must *specify the remote server as the "first server"* or else the command will fail with the following message:

```
There is an error with the certificate presented by the server.
```

This is the case even when you trust the unrecognized certificate interactively "for this session only."

- Run the **dsreplication** command interactively by starting it without arguments.

  Adapt the following example as appropriate for your deployment:

```
$ dsreplication
What do you want to do?

    1)  Configure Replication
    2)  Unconfigure Replication
    3)  Initialize Replication on one Server
    4)  Initialize All Servers
    5)  Pre External Initialization
    6)  Post External Initialization
    7)  Display Replication Status
    8)  Purge Historical
    9)  Re-synchronizes the change-log changenumber on one server with the
        change-log changenumber of another.

    c)  cancel

Enter choice: 1


>>>> Specify server administration connection parameters for the first server

Directory server hostname or IP address [local.example.com]: remote.example.com

Directory server administration port number [4444]:

How do you want to trust the server certificate?

    1)  Automatically trust
    2)  Use a truststore
    3)  Manually validate

Enter choice [3]:

Global Administrator User ID, or bind DN if no Global Administrator is defined
[admin]: cn=Directory Manager

Password for user 'cn=Directory Manager':

Server Certificate:



User DN  : CN=remote.example.com, O=Administration Connector RSA
Self-Signed Certificate
Validity : From '<start-date>'
           To '<end-date>'
Issuer   : CN=remote.example.com, O=Administration Connector RSA
Self-Signed Certificate


Do you trust this server certificate?

    1)  No
    2)  Yes, for this session only
    3)  Yes, also add it to a truststore
    4)  View certificate details
```

```
Enter choice [2]:
Replication port for the first server (the port must be free) [8989]:

Do you want replication to use encrypted communication when connecting to
replication port 8989 on the first server? (yes / no) [no]: yes



>>>> Specify server administration connection parameters for the second server

Directory server hostname or IP address [local.example.com]: local.example.com

Directory server administration port number [4444]:

How do you want to trust the server certificate?

    1)  Automatically trust
    2)  Use a truststore
    3)  Manually validate

Enter choice [3]:

Global Administrator User ID, or bind DN if no Global Administrator is defined
[admin]: cn=Directory Manager

Password for user 'cn=Directory Manager':
Replication port for the second server (the port must be free) [8989]:

Do you want replication to use encrypted communication when connecting to
replication port 8989 on the second server? (yes / no) [no]: yes


Global Administrator must be created.
You must provide the credentials of the Global Administrator that will be
created to manage the server instances that are being replicated.
Global Administrator User ID [admin]:

Global Administrator Password:

Confirm Password:


You must choose at least one Base DN to be replicated.
Replicate base DN dc=example,dc=com? (yes / no) [yes]:

...
Replication has been successfully configured.  Note that for replication to work
you must initialize the contents of the base DNs that are being replicated
(use dsreplication initialize to do so).
...
```

Initializing Replicas

Initializing replication consists of bringing a directory server up to date with the latest changes. The aim is to get the directory server:

- The latest set of directory data.
- The latest state information (a pointer to the latest update).

How you initialize replication depends on your situation.

In deployments where all replicas are on the same LAN and network bandwidth is not an issue, initializing replication over the network is generally the easiest option.

In deployments with servers replicating over the WAN, consider the following points:

- DS backup archives and LDIF files are fully portable from one replica to another, even across operating systems.
- It may be quicker to transfer large files for initializing replication, such as backup archives or LDIF, than to rely on the replication protocol to transfer data.

  For example, if individual WAN connections are slow, but you have plenty of WAN bandwidth, consider splitting large files, transferring the parts across multiple WAN connections, and then merging the parts after the transfer.
- Backup archives include indexes and also garbage data not yet collected by cleaner threads.

  When you initialize replication over the network or starting from an LDIF export, you only transfer the data. The replica builds the indexes locally.

  In deployments with many indexes, especially expensive indexes such as substring or JSON attribute indexes, it may be faster to transfer the data as LDIF or over the replication protocol.
- Online initialization has a performance impact on the replica, as the replica must replay many changes at the same time it serves other client requests. This extra throughput can cause response times to rise.

  You can avoid this by stopping the replica and restoring from backup or importing LDIF to prepare initialization while the server is offline.

"How To Initialize Replication" suggests appropriate procedures for specific situations.

How To Initialize Replication

Your Situation	See...
Small data set, fast network Evaluating DS software	"To Initialize Replication Over the Network"

Your Situation	See...
Developing a deployment solution	
Small data set, fast network Adding replica to existing topology	"To Initialize a Single New Replica Over the Network"
Medium to large data set (> 500,000 entries), new directory service	"To Initialize All Servers From the Same LDIF"
Medium to large data set (> 500,000 entries), existing directory service	"To Initialize a Replica From Backup Files"
Broken data set	"To Restore All Replicas to a Known State"

Important

Initialize replication after you have finished configuring replication.

If you attempt to run a **dsreplication configure** command at the same time a **dsreplication initialize** command is in progress, the **dsreplication configure** command results in an error such as the following:

```
Unwilling to Perform: ... Cannot change the configuration while a total update is in progress
```

To Initialize Replication Over the Network

The simplest approach for the directory administrator is to initialize replicas through the replication protocol's total update capability:

1. Make sure you have configured replication for all servers.

   See "Configuring Replication" for instructions.

2. Initialize replication.

   The following command uses **dsreplication initialize-all** to overwrite the data in all replicas with the data from the replica running the command:

   ```
   $ dsreplication \
    initialize-all \
    --adminUID admin \
    --adminPassword password \
    --baseDN dc=example,dc=com \
    --hostname opendj.example.com \
    --port 4444 \
    --trustAll \
    --no-prompt
   ```

   *Only use **initialize-all** when initializing all replicas.* To initialize a single replica, use **dsreplication initialize** instead as shown in "To Initialize a Single New Replica Over the Network".

To Initialize a Single New Replica Over the Network

Follow these steps when adding and initializing a single replica over the network using the replication protocol:

1. Configure replication on the new directory server.

   See "Configuring Replication" for instructions.

2. Initialize replication.

   The following command uses **dsreplication initialize** to overwrite the data in the new destination replica with the data from the existing source replica:

```
$ dsreplication \
 initialize \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --hostSource opendj.example.com \
 --portSource 4444 \
 --hostDestination new-replica.example.com \
 --portDestination 4444 \
 --trustAll \
 --no-prompt
```

To Initialize All Servers From the Same LDIF

Important

If you aim to return to a previous state of the data, or to initialize replicas with LDIF from a non-replicated environment, follow the steps in "To Restore All Replicas to a Known State" instead.

Use this procedure to initialize a new directory service when you have the same large LDIF available on all directory servers' local disks:

1. Configure replication for all servers.

   See "Configuring Replication" for instructions.

2. (Optional) If you have not already done so, enable data confidentiality as described in "Encrypting Directory Data" and "To Encrypt External Change Log Data".

3. Import the same LDIF on all servers as described in "To Import LDIF Data".

   Do not yet accept updates to the directory data. "Read-Only Replicas" shows how to prevent replicas from accepting updates from client applications.

4. Allow updates from client applications by setting `writability-mode:enabled` using a command like the one shown in "Read-Only Replicas".

To Initialize a Replica From Backup Files

You can create a new replica or initialize an outdated server from a backup of an existing replica:

1. If you are creating a new replica, install the new directory server.
2. If you are creating a new replica, configure replication:

```
$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 opendj.example.com \
 --port1 4444 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --replicationPort1 8989 \
 --host2 new-replica.example.com \
 --port2 4444 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --replicationPort2 8989 \
 --trustAll \
 --no-prompt
```

   Do not use the **dsreplication initialize** command.

3. Obtain fresh backup files from an existing server, as described in "Backing Up Directory Data".

   In this context, "fresh" means that after you restore the backup on the new server or outdated replica and restart the server, the restored data will be newer than the replication purge delay (default: 3 days).

   At this time, other replicas in the topology continue to accept updates.

4. On the new server, restore the database from the backup archive as described in "To Restore a Replica".

   As long as the restore operation completes and the directory server processes replication updates before the replication purge delay runs out, replication replays all changes made on other servers after you created the backup.

To Restore All Replicas to a Known State

DS replication is designed to make directory data converge across all replicas in a topology. Directory replication mechanically applies new changes to ensure that replicated data is the same everywhere, with newer changes taking precedence over older changes.

When you restore older backup data, for example, directory replication applies newer changes to the older data. This behavior is a good thing when the newer changes are correct.

This behavior can be problematic in the following cases:

- A bug or serious user error results in unwanted new changes that are hard to fix.
- The data in a test or proof-of-concept environment must regularly be reinitialized to a known state.

The **dsreplication** command has the following subcommands that let you reinitialize directory data, preventing replication from replaying changes that occurred before reinitialization:

- The **dsreplication pre-external-initialization** command removes the setting for the *generation ID* across the topology for a specified base DN. The generation ID is an internal-use identifier that replication uses to determine what changes to apply. This halts replication.
- The **dsreplication post-external-initialization** command sets a new generation ID across the topology, effectively resuming replication.

Caution

The steps in this procedure reinitialize the replication changelog, eliminating the history of changes that occurred before replication resumed. The replication changelog is described in "Change Notification For Your Applications". Applications that depend on the changelog for change notifications must be reinitialized after this procedure is completed.

1. (Optional) Prevent changes to the affected data during the procedure, as such changes are lost for the purposes of replication.

   For example, make each replica read-only as described in "Read-Only Replicas".

2. On a single server in the topology, run the **dsreplication pre-external-initialization** command for the base DN holding the relevant data, as shown in the following example:

   ```
   $ dsreplication \
    pre-external-initialization \
    --adminUID admin \
    --adminPassword password \
    --hostname opendj.example.com \
    --port 4444 \
    --baseDN dc=example,dc=com \
    --trustAll \
    --no-prompt
   ```

   Replication halts as the command takes effect.

   Changes made at this time are not replicated, even after replication resumes.

3. On each server in the topology, restore the data in the topology to the known state in one of the following ways:

- Import the data from LDIF as described in "To Import LDIF Data".
- Restore the data from backup as described in "To Restore a Standalone Server".

4. On a single server in the topology, run the **dsreplication post-external-initialization** command for the base DN holding the relevant data, as shown in the following example:

```
$ dsreplication \
 post-external-initialization \
 --adminUID admin \
 --adminPassword password \
 --hostname opendj.example.com \
 --port 4444 \
 --baseDN dc=example,dc=com \
 --trustAll \
 --no-prompt
```

   Replication resumes as the command takes effect.

5. (Optional) If you made replicas read-only, make them read-write again by setting `writability-mode:enabled`.

Setting Disk Space Thresholds For Replication Changelog Databases

Replication servers record changes in changelog database files. By default, changelog database files are located under the `/path/to/opendj/changelogDb` directory. The changelog database is internal to replication, but there is a public interface described in "Change Notification For Your Applications". The public configuration properties are part of the replication server configuration.

Important

Do not compress, tamper with, or otherwise alter changelog database files directly unless specifically instructed to do so by a qualified ForgeRock technical support engineer. External changes to changelog database files can render them unusable by the server.

Replication server configuration objects have changelog database properties, `disk-low-threshold` and `disk-full-threshold`. When available disk space falls below `disk-low-threshold`, the replication server triggers a warning alert notification to indicate that you must free some disk space. When available space falls below `disk-full-threshold`, the replication server triggers another warning alert notification, and *disconnects from the replication topology*. Connected directory servers fail over to another replication server until disk space rises above the `disk-full-threshold` again.

Set the thresholds high enough to allow time to react after the initial alert.

The following example sets `disk-low-threshold` to 10 GB and `disk-full-threshold` to 5 GB:

```
$ dsconfig \
 set-replication-server-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set "disk-low-threshold:10 GB" \
 --set "disk-full-threshold:5 GB" \
 --trustAll \
 --no-prompt
```

The properties `disk-low-threshold` and `disk-full-threshold` are listed as *advanced* properties. To examine their values with the **dsconfig** command, use the `--advanced` option when running the command, or change the option while using the **dsconfig** command in interactive mode.

Stopping Replication

How you stop replication depends on whether the change is meant to be temporary or permanent.

To Stop Replication Temporarily For a Replica

If you must stop a server from replicating temporarily, you can do so by using the **dsreplication suspend** and **dsreplication resume** commands.

Warning

Do not allow modifications on the replica for which replication is temporarily stopped. No record of such changes is kept, and the changes cause replication to diverge.

For details, see "Read-Only Replicas".

1. Run the **dsreplication suspend** command:

```
$ dsreplication \
 suspend \
 --hostname replica.example.com \
 --port 4444 \
 --adminUid admin \
 --adminPassword password \
 --trustAll \
 --no-prompt
```

2. (Optional) Run the **dsreplication resume** command:

```
$ dsreplication \
 resume \
 --hostname replica.example.com \
 --port 4444 \
 --adminUid admin \
 --adminPassword password \
 --trustAll \
 --no-prompt
```

To Stop Replication Permanently For a Replica

If you need to stop a server from replicating permanently, for example, when retiring a server, use the **dsreplication unconfigure** command.

Important

When you set **dsreplication** hostname options, such as `--hostname`, `--host1`, and `--host2`, make sure to always use either FQDNs or IP addresses. Use whichever are most stable and well-known to servers in your environment. (Never use local-only names or addresses, such as `localhost`, for production systems as they do not uniquely identify the server to other servers.)

Make sure that you use the same hostname value that was provided at installation time to the **setup** command. If you do not know the original hostname used at installation time, read the entry with DN `cn=DIGEST-MD5, cn=SASL Mechanisms,cn=config`. Its `ds-cfg-server-fqdn` attribute holds the original hostname.

Furthermore, always use the same hostname value for the same server each time you invoke the **dsreplication** command. The server uses the values you specify in its configuration. When you use one value to configure replication and another to unconfigure replication, the latter operation fails on remote servers, leaving the system in a broken state.

If you do not know the original hostname values used with the **dsreplication** command, read the entry with DN `cn=all-servers,cn=Server Groups,cn=admin data`. Its `uniqueMember` attribute holds the hostname values.

1. Stop replication using the **dsreplication unconfigure** command:

```
$ dsreplication \
 unconfigure \
 --hostname new-replica.example.com \
 --port 4444 \
 --adminUID admin \
 --adminPassword password \
 --unconfigureAll \
 --trustAll \
 --no-prompt
```

The **dsreplication unconfigure** command completely removes the replication configuration information from the server.

2. (Optional) To restart replication for the server, run the **dsreplication configure** and **dsreplication initialize** commands again.

Standalone Replication Servers

DS replication is designed to be easy to implement in environments with a few servers, and scalable in environments with many servers. You can configure the replication service on each DS directory server in your deployment, for example, to limit the number of servers you deploy.

In a large deployment, you can use standalone replication servers—servers that do nothing but relay replication messages—to configure (and troubleshoot) the replication service separately from the directory service. You need only a few standalone replication servers publishing changes to serve many directory servers subscribed to the changes. Furthermore, replication is designed so a directory server needs to connect only to the nearest replication server in order to to replicate with all servers in the topology. Only the standalone replication servers participate in fully meshed replication.

All replication servers in a topology are connected to all other replication servers. Directory servers are connected only to one replication server at a time, and their connections should be to replication servers on the same LAN. Therefore, the total number of replication connections, $Total_{conn}$, is expressed as follows:

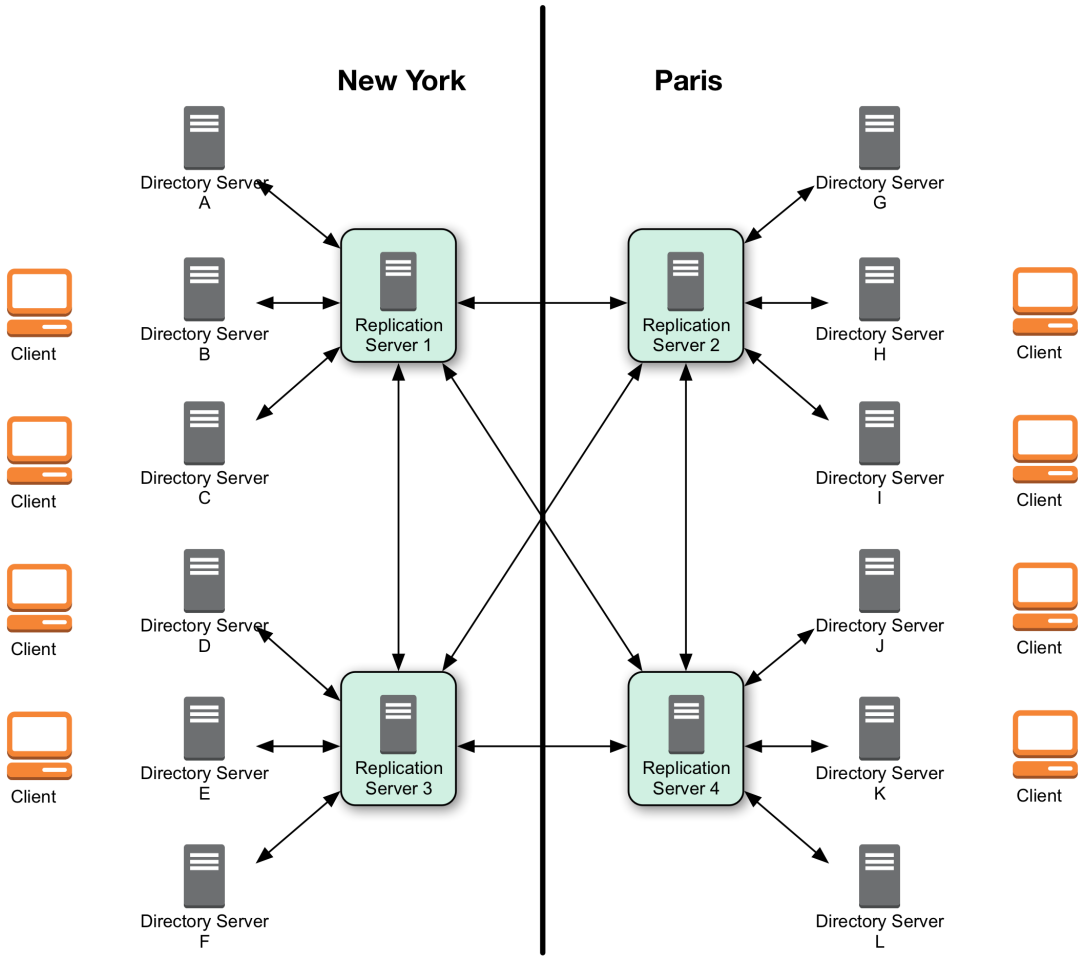
$$Total_{conn} = (N_{RS} * (N_{RS}-1))/2 + N_{DS} \quad (8.1)$$

Here, N_{RS} is the number of replication servers, and N_{DS} is the number of standalone directory servers.

In other words, if you have only three servers, then $Total_{conn}$ is three with no standalone servers. However, if you have two data centers, and need 12 directory servers, then with no standalone directory servers $Total_{conn}$ is (12 * 11)/2 or 66. Yet, with four standalone replication servers, and 12 standalone directory servers, $Total_{conn}$ is (4 * 3)/2 + 12, or 18. Only four of those connections extend over the WAN. (By running four directory servers that also run replication servers and eight standalone directory servers, you reduce the number of replication connections to 14 for 12 replicas.)

This is shown in "Deployment For Multiple Data Centers".

Deployment For Multiple Data Centers



To Use a Standalone Replication Server

This procedure uses a standalone replication server to handle the replication traffic between two standalone directory servers.

In a production deployment, set up additional standalone replication servers to avoid a single point of failure.

Follow these steps:

1. Set up a standalone replication server, for example `rs-only.example.com`, as described in "*Installing a Replication Server*" in the *Installation Guide*.

2. Set up two standalone directory servers, for example `opendj.example.com` and `replica.example.com`, as described in "*Installing a Directory Server*" in the *Installation Guide*.

3. Configure replication with the `--noReplicationServer` or `--onlyReplicationServer` option:

```
$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 opendj.example.com \
 --port1 4444 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --noReplicationServer1 \
 --host2 rs-only.example.com \
 --port2 4444 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --replicationPort2 8989 \
 --onlyReplicationServer2 \
 --trustAll \
 --no-prompt

$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 replica.example.com \
 --port1 4444 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --noReplicationServer1 \
 --host2 rs-only.example.com \
 --port2 4444 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --replicationPort2 8989 \
 --onlyReplicationServer2 \
 --trustAll \
 --no-prompt
```

4. Initialize replication from one of the directory servers, for example `opendj.example.com`:

```
$ dsreplication \
 initialize-all \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --hostname opendj.example.com \
 --port 4444 \
 --trustAll \
 --no-prompt
```

Standalone Directory Server Replicas

When you configure replication for a directory server, you can give the directory server the capability to handle replication traffic as well. As described in "Standalone Replication Servers", DS servers can also be configured to handle only replication traffic.

Alternatively you can configure a directory server to connect to a remote replication server of either variety, but to remain only a directory server itself. This sort of standalone directory server replica is shown in "Deployment For Multiple Data Centers".

Furthermore, you can make this standalone directory server replica read-only for client applications, accepting only replication updates.

- "To Set Up a Standalone Directory Server Replica"
- "To Transform a Directory Server/Replication Server Into a Standalone Directory Server"

To Set Up a Standalone Directory Server Replica

The following steps show how to configure the server as a standalone, directory server-only replica of an existing replicated directory server:

1. Set up replication between other servers.
2. Install the directory server without configuring replication, but creating at least the base entry to be replicated.
3. Configure replication with the appropriate `--noReplicationServer` option:

```
$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 opendj.example.com \
 --port1 4444 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --replicationPort1 8989 \
 --host2 ds-only.example.com \
 --port2 4444 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --noReplicationServer2 \
 --trustAll \
 --no-prompt
```

Here the existing server is both directory server and replication server. If the existing server is a standalone replication server, then also use the appropriate `--onlyReplicationServer` option.

4. Initialize data on the new directory server replica:

```
$ dsreplication \
 initialize \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --hostSource opendj.example.com \
 --portSource 4444 \
 --hostDestination ds-only.example.com \
 --portDestination 4444 \
 --trustAll \
 --no-prompt
```

5. If you want to make the directory server replica read-only for client application traffic, see "Read-Only Replicas".

To Transform a Directory Server/Replication Server Into a Standalone Directory Server

If you configured your replicas as both directory servers and replication servers, you can unconfigure the replication server component, leaving the replica as a standalone directory server:

1. Before you start, make sure the replica is correctly configured to replicate with other servers.

2. Unconfigure the replication server on the replica to transform it into a standalone directory server:

```
$ dsreplication \
 unconfigure \
 --unconfigureReplicationServer \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDn uid=admin \
 --adminPassword password \
 --trustAll \
 --no-prompt
```

Replication Groups

Replication lets you define groups so that replicas communicate first with replication servers in the group before going to replication servers outside the group. Groups are identified with unique group IDs.

Replication groups are designed for deployments across multiple data centers, where you aim to focus replication traffic on the LAN rather than the WAN. In multi-data center deployments, group nearby servers together.

To Set Up Replication Groups

For each group, set the appropriate group ID for the topology on both the replication servers and the directory servers.

The example commands in this procedure set up two replication groups, each with a replication server and a directory server. The directory servers are `opendj.example.com` and `replica.example.com`. The replication servers are `rs.example.com` and `rs2.example.com`. In a full-scale deployment, you would have multiple servers of each type in each group. All directory servers and replication servers in a data center would belong to the same group.

1. Pick a group ID for each group.

   The default group ID is `default`. For mixed topologies with replicas running older DS versions that support only numeric group IDs, this is considered equivalent to `1`.

2. Set the group ID for each group by replication domain on the directory servers:

```
$ dsconfig \
 set-replication-domain-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name "dc=example,dc=com" \
 --set group-id:US-East \
 --trustAll \
 --no-prompt

$ dsconfig \
 set-replication-domain-prop \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name "dc=example,dc=com" \
 --set group-id:US-West \
 --trustAll \
 --no-prompt
```

3. Set the group ID for each group on the replication servers:

```
$ dsconfig \
 set-replication-server-prop \
 --hostname rs.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set group-id:1 \
 --trustAll \
 --no-prompt

$ dsconfig \
 set-replication-server-prop \
 --hostname rs2.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set group-id:2 \
 --trustAll \
 --no-prompt
```

Read-Only Replicas

By default all directory servers in a replication topology are read-write. You can, however, choose to make replicas take updates only from the replication protocol, and refuse updates from client applications:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set writability-mode:internal-only \
 --trustAll \
 --no-prompt
```

Subtree Replication

DS servers can perform subtree replication. For example, they can replicate `ou=People,dc=example,dc=com`, but not the rest of `dc=example,dc=com`, by putting the subtree in a separate backend from the rest of the suffix.

In this example, you might have an `exampleData` backend containing everything in `dc=example,dc=com` except `ou=People,dc=example,dc=com`, and a separate `peopleData` backend for `ou=People,dc=example,dc=com`. You can then configure DS servers to replicate `ou=People,dc=example,dc=com` in a separate topology. You cannot, however, replicate `dc=example,dc=com` on the same servers due to limitations described in "Replication Per Suffix".

For details, see "Splitting Data Across Multiple Backends".

Fractional Replication

DS servers can perform fractional replication. For fractional replication, you specify the attributes to include in the replication process, or the attributes to exclude from the replication process.

You set fractional replication configuration as `fractional-include` or `fractional-exclude` properties of a replication domain. When you include attributes, the attributes that are required on the relevant object classes are also included, whether you specify them or not. When you exclude attributes, the excluded attributes must be optional attributes for the relevant object classes. Fractional replicas still respect schema definitions.

Fractional replication filters objects at the replication server level. Each attribute must remain available on at least one replica in the topology. Fractional replication is not designed to exclude the same attribute on every replica in a topology. When you configure a replica to exclude an attribute, the directory server checks that the attribute is never added to the replica as part of any LDAP operation. As a result, if you exclude the attribute everywhere, it can never be added anywhere.

When using fractional replication, initialize replication as you would normally. You cannot create a full replica, however, from a replica with only a subset of the data. If you must prevent data from

being replicated across a national boundary, for example, split the replication server that handles updates from the directory servers as described in "To Use a Standalone Replication Server".

For example, you might configure an externally facing fractional replica to include only some `inetOrgPerson` attributes:

```
$ dsconfig \
 set-replication-domain-prop \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name "dc=example,dc=com" \
 --set fractional-include:inetorgperson:cn,givenname,mail,mobile,sn,telephonenumber \
 --trustAll \
 --no-prompt
```

As another example, you might exclude a custom attribute called `sessionToken` from being replicated:

```
$ dsconfig \
 set-replication-domain-prop \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name "dc=example,dc=com" \
 --set fractional-exclude:*:sessionToken \
 --trustAll \
 --no-prompt
```

The latter example only has an effect if you first define a `sessionToken` attribute in the directory server schema.

Choosing the Listen Address for Replication

When configuring a replication server on a multi-homed system with multiple IP addresses, you can specify which `listen-address` or addresses to use. By default, the replication server listens on all addresses.

The following example sets the replication server to listen only on `192.168.0.10`:

```
$ dsconfig \
 set-replication-server-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set listen-address:192.168.0.10 \
 --trustAll \
 --no-prompt
```

Change Notification For Your Applications

Some applications require notification when directory data updates occur. For example, an application might need to sync directory data with another database, or the application might need to kick off other processing when certain updates occur.

In addition to supporting persistent search operations, DS replication servers provide an external change log mechanism to allow applications to be notified of changes to directory data.

Important

Do not compress, tamper with, or otherwise alter changelog database files directly unless specifically instructed to do so by a qualified ForgeRock technical support engineer. External changes to changelog database files can render them unusable by the server. By default, changelog database files are located under the `/path/to/opendj/changelogDb` directory.

This section includes the following procedures:

- "To Enable the External Change Log"
- "To Encrypt External Change Log Data"
- "To Use the External Change Log"
- "To Allow a User to Read the Change Log"
- "To Include Unchanged Attributes in the External Change Log"
- "To Align Draft Change Numbers"
- "To Disable Change Number Indexing"

To Enable the External Change Log

DS servers that have a replication server port and an LDAP port publish an external changelog over LDAP:

Ports Configured	Examples	Notes
LDAP or LDAPS port	1389, 1636	LDAP client applications use the LDAP or LDAPS port to read changelog data. A standalone replication server may not have an LDAP or LDAPS port configured.
Replication port	8989	Servers with replication ports maintain a changelog for their own use. The changelog is exposed over LDAP under the base DN, `cn=changelog`. Standalone directory servers do not maintain a changelog by default.

1. Make sure an LDAP or LDAPS port is configured so that LDAP client applications can read the changelog.
2. Depending on the server configuration, enable the changelog in one of the following ways:
   - For servers with replication ports, make sure replication is properly configured.

     The following example shows how the directory superuser can read the changelog:

     ```
     $ ldapsearch \
      --hostname opendj.example.com \
      --port 1389 \
      --bindDN "cn=Directory Manager" \
      --bindPassword password \
      --baseDN cn=changelog \
      --searchScope base \
      "(&)"
     dn: cn=changelog
     objectClass: top
     objectClass: container
     cn: changelog
     ```

   - For standalone directory servers without replication ports, manually enable the changelog:

     a. Create a replication server configuration entry as in the following example:

     ```
     $ dsconfig \
      create-replication-server \
      --hostname opendj.example.com \
      --port 4444 \
      --bindDN "cn=Directory Manager" \
      --bindPassword password \
      --provider-name "Multimaster Synchronization" \
      --set replication-port:8989 \
      --set replication-server-id:2 \
      --trustAll \
      --no-prompt
     ```

     b. Create a replication domain configuration entry as in the following example:

```
$ dsconfig \
 create-replication-domain \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name "Example.com" \
 --set base-dn:dc=example,dc=com \
 --set replication-server:opendj.example.com:8989 \
 --set server-id:3 \
 --trustAll \
 --no-prompt
```

3. Make sure the user who needs to read changelog data has the `changelog-read` privilege, and has access to read entries under `cn=changelog`.

   For details, see "To Allow a User to Read the Change Log".

To Encrypt External Change Log Data

DS servers do not encrypt external change log data by default. This means that any user with system access to read directory files can potentially access external change log data in cleartext.

In addition to preventing read access by other users as described in "Setting Up a System Account for a Server", you can configure confidentiality for external change log data. When confidentiality is enabled, the server encrypts change log records before storing them.

Important

Encrypting stored directory data does not prevent it from being sent over the network in the clear.

Apply the suggestions in "*Securing Network Connections*" in the *Security Guide* to protect data sent over the network.

DS servers encrypt data using a symmetric key that is stored with the server configuration. The symmetric key is encrypted in turn with the server's public key that is also stored with the server configuration. When multiple servers are configured to replicate data as described in "Configuring Replication Settings", the servers replicate the keys as well, allowing any server replica to decrypt the data.

Encrypting and decrypting data comes with costs in terms of cryptographic processing that reduces throughput and of extra space for larger encrypted values. In general, tests with default settings show that the cost of enabling confidentiality can be quite modest, but your results can vary based on your systems and on the settings used for `cipher-transformation` and `cipher-key-length`. Make sure you test your deployment to qualify the impact of confidentiality before enabling it in production.

Follow this procedure to enable confidentiality:

1. Before you enable confidentiality on a replication server for the external change log data, first enable confidentiality for data stored in directory backends.

For details, see "Encrypting Directory Data".

2. Enable backend confidentiality with the default encryption settings as shown in the following example:

```
$ dsconfig \
 set-replication-server-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set confidentiality-enabled:true \
 --no-prompt \
 --trustAll
```

   Encryption applies to the entire change log regardless of the confidentiality settings for each domain.

   After confidentiality is enabled, new change log records are encrypted. DS servers do not rewrite old records in encrypted form.

3. (Optional) If necessary, adjust additional confidentiality settings.

   Use the same cipher suite for external change log confidentiality as was used to configure data confidentiality.

   The default settings for confidentiality are `cipher-transformation: AES/CBC/PKCS5Padding` and `cipher-key-length: 128`. This means the algorithm is the Advanced Encryption Standard (AES), the cipher mode is Cipher Block Chaining (CBC), and the padding is PKCS#5 padding as described in RFC 2898: PKCS #5: Password-Based Cryptography Specification. The syntax for the `cipher-transformation` is *`algorithm/mode/padding`*, and all three must be specified. When the algorithm does not require a mode, use `NONE`. When the algorithm does not require padding, use `NoPadding`. Use of larger `cipher-key-length` values can require that you install JCE policy files such as those for unlimited strength, as described in "Using Unlimited Strength Cryptography" in the *Security Guide*.

To Use the External Change Log

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

The external change log mechanism uses an LDAP control with OID `1.3.6.1.4.1.26027.1.5.4` to allow the exchange of cookies for the client application to bookmark the last changes seen, and then start reading the next set of changes from where it left off on the previous request.

This procedure shows the client reading the change log as `cn=Directory Manager`. Make sure your client application reads the changes with sufficient access and privileges to view all the changes it needs to see.

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

   In this example, two changes appear in the changelog:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "1.3.6.1.4.1.26027.1.5.4:false" \
 "(&)" \
 changes changeLogCookie targetDN
# Public changelog exchange control(1.3.6.1.4.1.26027.1.5.4): <COOKIE1>
dn: replicationCSN=<CSN1>,dc=example,dc=com,cn=changelog
changes::
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldyBkZXNjcmlwdGlvbgotCnJlcGxhY2U6IG1vZGlmaWVyc05hbWUKbW9kaWZpZX
targetDN: uid=bjensen,ou=People,dc=example,dc=com
changeLogCookie: <COOKIE1>

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

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

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

   Notice the `changeLogCookie` value, which has the form *`base-dn:CSN`*.

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

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "1.3.6.1.4.1.26027.1.5.4:false:$COOKIE1" \
 "(&)" \
 changes changeLogCookie targetDN
# Public changelog exchange control(1.3.6.1.4.1.26027.1.5.4): <COOKIE2>
dn: replicationCSN=<CSN2>,dc=example,dc=com,cn=changelog
changes::
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldywgaW1wcm92ZWQgZGVzY3JpcHRpb24KLQpyZXBsYWNlOiBtb2RpZmllcnNOYW
targetDN: uid=bjensen,ou=People,dc=example,dc=com
changeLogCookie: <COOKIE2>
```

The following command decodes the changes:

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

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

To Allow a User to Read the Change Log

For a user to read the changelog, the user must have access to read, search, and compare changelog attributes, might have access to use the control to read the external changelog, and must have the `changelog-read` privilege.

1. Give the user access to read and search the changelog.

   The following example adds two global ACIs. The first ACI gives `My App` read access to root DSE attributes that hold information about the changelog. The second ACI gives `My App` read access to the changelog data:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(target=\"ldap:///\")\
(targetattr=\"changeLog||firstChangeNumber||lastChangeNumber||lastExternalChangeLogCookie\")\
(version 3.0; acl \"My App can access changelog attributes on root DSE\"; \
allow (read,search,compare) \
userdn=\"ldap:///cn=My App,ou=Apps,dc=example,dc=com\";)" \
 --add global-aci:"(target=\"ldap:///cn=changelog\")(targetattr=\"*||+\")\
(version 3.0; acl \"My App can access cn=changelog\"; \
allow (read,search,compare) \
userdn=\"ldap:///cn=My App,ou=Apps,dc=example,dc=com\";)" \
 --trustAll \
 --no-prompt
```

The IDM liveSync feature requires access to the root DSE attributes `changeLog`, `firstChangeNumber`, and `lastChangeNumber`.

2. Give the user access to use the public changelog exchange control.

   The following example adds a global ACI to give `My App` access to use the control:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"1.3.6.1.4.1.26027.1.5.4\")\
(version 3.0; acl \"My App control access\"; \
allow (read) \
userdn=\"ldap:///cn=My App,ou=Apps,dc=example,dc=com\";)" \
 --trustAll \
 --no-prompt
```

3. Give the user the `changelog-read` privilege:

```
$ cat changelog-read.ldif
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: changelog-read

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

4. Check that the user can read the changelog:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "1.3.6.1.4.1.26027.1.5.4:false" \
 "(&)" \
 changes changeLogCookie targetDN
```

To Include Unchanged Attributes in the External Change Log

As shown above, the changes returned from a search on the external change log include only what was actually changed. If you have applications that need additional attributes published with every change log entry, regardless of whether or not the attribute itself has changed, then specify those using `ecl-include` and `ecl-include-for-deletes`.

1. Set the attributes to include for all update operations with `ecl-include`:

```
$ dsconfig \
 set-external-changelog-domain-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name dc=example,dc=com \
 --set ecl-include:"@person" \
 --trustAll \
 --no-prompt
```

2. Set the attributes to include for deletes with `ecl-include-for-deletes`:

```
$ dsconfig \
 set-external-changelog-domain-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --domain-name dc=example,dc=com \
 --add ecl-include-for-deletes:"*" \
 --add ecl-include-for-deletes:"+" \
 --trustAll \
 --no-prompt
```

To Align Draft Change Numbers

The external change log can be used by applications that follow the *Internet-Draft: Definition of an Object Class to Hold LDAP Change Records*, and that cannot use change log cookies shared across the replication topology. Nothing special is required to get the objects specified for this legacy format, but there are steps you must perform to align change numbers across replicas.

Change numbers described in the Internet-Draft are simple numbers, not cookies. When change log numbers are aligned across replicas, applications fail over from one replica to another when necessary.

If you do not align the change numbers, each server keeps its own count. The same change numbers can refer to different changes on different replicas.

For example, if you install a new replica and initialize replication from an existing server, the last change numbers are likely to differ. The following example shows different last change numbers for an existing server and for a new replica that has just been initialized from the existing replica:

```
$ ldapsearch \
 --hostname existing.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN "" \
 --searchScope base \
 "(&)" lastChangeNumber
dn:
lastChangeNumber: 285924


Result Code:  0 (Success)

$ ldapsearch \
 --hostname new.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN "" \
 --searchScope base \
 "(&)" lastChangeNumber
dn:
lastChangeNumber: 198643


Result Code:  0 (Success)
```

When you add a new replica to an existing topology, follow these steps to align the change numbers with those of an existing server.

These steps can also be used at any time to align the change numbers:

1. Make sure that the new replica has the same replication configuration as the existing replica.

   Specifically, both replicas must replicate the same suffixes in order for the change number calculations to be the same on both replicas. If the suffix configurations differ, the change numbers cannot be aligned.

2. (Optional) If you must start the new replica's change numbering from a specific change, determine the `changeNumber` to use.

   The `changeNumber` must be from a change that has not yet been purged according to the replication purge delay, which by default is three days.

3. Using the **dsreplication** command installed with the new replica, reset the change number on the new replica to the change number from the existing replica.

   The following example does not specify the change number to use. By default, the new replica uses the last change number from the existing replica:

   ```
   $ dsreplication \
    reset-change-number \
    --adminUID admin \
    --adminPassword password \
    --hostSource existing.example.com \
    --portSource 4444 \
    --hostDestination new.example.com \
    --portDestination 4444 \
    --trustAll \
    --no-prompt
   Change-log change number reset task has finished successfully.
   ...
   ```

   At this point, the new replica's change log starts with the last change number from the existing replica. Earlier change numbers are no longer present in the new replica's change log.

To Disable Change Number Indexing

By default, a replication server indexes change numbers for replicated user data. This allows legacy applications to get update notifications by change number, as described in "To Align Draft Change Numbers". Indexing change numbers requires additional CPU, disk accesses and storage, so it should not be used unless change number-based browsing is required.

Disable change number indexing if it is not needed:

- Adjust the replication server settings appropriately for your deployment:
  - If applications need change notifications, but use changelog cookies rather than change numbers, set `changelog-enabled:enabled-cookie-mode-only in the Configuration Reference` on each server after enabling replication.

    The replication server no longer indexes change numbers, and so has less work to do.

- If no applications need change number-based notifications, set `changelog-enabled:disabled in the *Configuration Reference*` on each server after enabling replication.

Recovering From User Error

This section covers how to restore accidentally deleted or changed data.

Changes to a replicated DS directory service are similar to those made with the Unix **rm** command, but with a twist. With the **rm** command, if you make a mistake you can restore your files from backup, and lose only the work done since the last backup. If you make a mistake with a update to the directory service however, then after you restore a server from backup, replication efficiently replays your mistake to the server you restored.

There is more than one way to recover from user error. None of the ways involve simply changing DS settings. All of the ways instead involve manually fixing mistakes.

Consider these alternatives:

- Encourage client applications to provide end users with undo capability if necessary. In this case, client applications take responsibility for keeping an undo history.
- Maintain a record of each update to the service, so that you can manually "undo" mistakes.

  You can use the external change log. A primary advantage to the external change log is that the change log is enabled with replication, and so it does not use additional space.

  See "Change Notification For Your Applications" for instructions on enabling, using, and configuring the external change log. In particular, see "To Include Unchanged Attributes in the External Change Log" for instructions on saving not only what is changed, but also all attributes when an entry is deleted.

  DS servers can also write to a file-based audit log, but the audit log does not help with a general solution in this case. The DS audit log records changes to the data. When you delete an entry however, the audit log does not record the entry before deletion. The following example shows the audit log records of some changes made to Barbara Jensen's entry:

```
# <datestamp>; conn=<number>; op=<number>
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: This is the description I want.
-
replace: modifiersName
modifiersName: cn=Directory Manager
-
replace: modifyTimestamp
modifyTimestamp: <timestamp>

# <datestamp>; conn=<number>; op=<number>
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: I never should have changed this!
-
replace: modifiersName
modifiersName: cn=Directory Manager
-
replace: modifyTimestamp
modifyTimestamp: <timestamp>

# <datestamp>; conn=<number>; op=<number>
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: delete
```

You can use these records to fix the mistaken update to the description, but the audit log lacks the information needed to restore Barbara Jensen's deleted entry.

For details about the audit log, see "Native LDAP Audit Logs".

- For administrative errors that involve directory data, if you have properly configured the external change log, then use it.

  If not, an alternative technique consists of restoring backup to a separate server not connected to the replication topology. (Do not connect the server to the topology as replication replays mistakes, too.) Compare data on the separate restored server to the live servers in the topology, and then fix the mistakes manually.

  A more drastic alternative consists of rebuilding the entire service from backup, by unconfiguring replication and restoring all servers from backup (or restoring one server and initializing all servers from that one). This alternative is only recommended in the case of a major error where you have a very fresh backup (taken immediately before the error), and no client applications are affected.

- For administrative configuration errors that prevent servers from starting, know that DS servers keep snapshots of the main configuration file, including `/path/to/opendj/var/config.ldif.startok` and the files in the `/path/to/opendj/var/archived-configs/` directory.

You can therefore compare the current configuration with the earlier configurations, and repair mistakes manually (avoiding trailing white space at the end of LDIF lines) while the server is down.

Resolving Replication Conflicts

Replication is eventually consistent by design. Rather than dealing with some sort of distributed transaction mechanism that would make write operations impossible during a network partition, replication supports basic write availability. Changes are applied locally and then replayed to remote replicas. This means it is possible to have conflicts, however. A *replication conflict* arises when incompatible changes are applied concurrently to multiple read-write replicas.

Two types of conflicts can occur: *modify conflicts* and *naming conflicts*. Modify conflicts involve concurrent modifications to the same entry. Naming conflicts involve other operations that affect the DN of the entry.

Replication can resolve modify conflicts and many naming conflicts automatically by replaying the changes in the correct order. To determine the relative order in which changes occurred, replicas retain historical information for each update. This information is stored in the target entry's `ds-sync-hist` operational attribute.

Replication resolves the following conflicts automatically using the historical information to order changes correctly:

- The attributes of a given entry are modified concurrently in different ways on different replicas.
- An entry is renamed on one replica while being modified on another replica.
- An entry is renamed on one replica while being renamed in a different way on another replica.
- An entry is deleted on one replica while being modified on another replica.
- An entry is deleted and another entry with the same DN added on one replica while the same entry is being modified on another replica.

Replication cannot resolve the following naming conflicts, so you must resolve them manually:

- Different entries that share the same DN are added concurrently on multiple replicas.
- An entry on one replica is moved (renamed) to use the same DN as a new entry concurrently added on another replica.
- A parent entry is deleted on one replica while a child entry is added or renamed concurrently on another replica.

When replication cannot resolve naming conflicts automatically, the server renames the conflicting entry using its `entryUUID` operational attribute. The resulting conflicting entry has a DN with the following form:

```
entryuuid=entryUUID-value+original-RDN,original-parent-DN
```

For each conflicting entry named in this way, resolve the conflict manually:

1. Get the conflicting entry or entries, and the original entry if available.

   The following example shows the result on one replica of a naming conflict when a `newuser` entry was added concurrently on two replicas:

   ```
   $ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=newuser)"
   dn: uid=newuser,ou=People,dc=example,dc=com
   objectClass: top
   objectClass: inetOrgPerson
   objectClass: organizationalPerson
   objectClass: person
   mail: newuser@example.com
   sn: User
   cn: New User
   ou: People
   description: Added on server 1
   uid: newuser

   dn: entryuuid=2f1b58c3-4bee-4215-88bc-88202a7bcb9d+uid=newuser,ou=People,dc=example,dc=com
   objectClass: top
   objectClass: inetOrgPerson
   objectClass: organizationalPerson
   objectClass: person
   mail: newuser@example.com
   sn: User
   cn: New User
   ou: People
   description: Added on server 2
   uid: newuser
   ```

2. If you want to preserve changes made on the conflicting entry or entries, apply the changes manually.

   The following example shows a modification to preserve both description values:

   ```
   $ cat fix-conflict.ldif
   dn: uid=newuser,ou=People,dc=example,dc=com
   changetype: modify
   add: description
   description: Added on server 2


   $ ldapmodify --port 1389 --bindDN "cn=Directory Manager" --bindPassword password fix-conflict.ldif
   ```

   For additional examples demonstrating how to apply changes to directory entries, see "Updating the Directory" in the *Developer's Guide*.

3. After making any necessary changes, manually delete the conflicting entry or entries.

The following example deletes the conflicting entry:

```
$ ldapdelete \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 entryuuid=2f1b58c3-4bee-4215-88bc-88202a7bcb9d+uid=newuser,ou=People,dc=example,dc=com
```

For additional examples, see "Deleting Entries" in the *Developer's Guide*.

Chapter 9

Backing Up and Restoring Data

This chapter covers management of directory data backup archives. For information on managing directory data in an interoperable format that is portable between LDAP directory servers, see "*Managing Directory Data*" instead. In this chapter you will learn to:

- Create backup archives
- Restore data from backup archives
- Back up and restore configuration files

DS directory servers let you back up and restore your data either in compressed, binary format, or in LDIF. This chapter shows you how to back up and to restore directory data from archives, and explains portability of backup archives, and how to back up server configuration data.

Important

As explained in "About Database Backends", cleanup processes applied by database backends can be writing data even when there are no pending client or replication operations. To back up a server using a file system snapshot, you must *stop the server before taking the snapshot*.

Before you implement a backup plan for a directory services deployment, devise a backup and restore strategy that meets your needs. Read the following articles for background when creating your plan:

- *How do I design and implement my backup and restore strategies for OpenDJ/DS (All versions)?*
- *FAQ: Backup and restore in OpenDJ/DS*

Backing Up Directory Data

When you install a directory server, the setup process creates a `bak/` directory as the default location for binary backups. When you create a backup, the `bak/backup.info` file contains information about the archive. This is acceptable if you have only one backend to back up. Each `backup.info` file only contains information about one backend, however. If you have more than one backend, then use a separate backup directory for each backend in order to have separate `backup.info` files for each backend ID.

Archives produced by the **backup** command only contain backups of the directory data. Snapshots of server configuration are found under the `var/` directory.

Important

The **backup** command can encrypt the backup data. It encrypts the data using a symmetric key that is stored with the server configuration. The symmetric key is encrypted in turn with the server's public key that is also stored with the server configuration. The mechanism is described in "How Secret Keys are Distributed" in the *Security Guide*.

In summary, when multiple servers are configured to replicate data as described in "*Managing Data Replication*", the servers replicate the keys as well, allowing any server replica to decrypt the backup data.

Even if you do not encrypt the backup archive, the directory data may include passwords encrypted using a reversible storage scheme, such as AES or Blowfish. The same symmetric key distribution mechanism based on replication is used for both reversible password storage schemes and encrypted backup.

If ever all servers in the replication topology are lost, new servers can no longer decrypt any encrypted backup files or reversible passwords.

To work around this limitation, maintain a file system backup of at least one server from each replication topology in your deployment. To recover from a disaster where all servers in the topology were lost, restore the server files from the file system backup, and start the restored server. Other new servers whose data you restore from encrypted backup can then obtain the decryption keys from the restored server as described in "To Restore a Replica".

This section includes the following procedures:

- "To Back Up Data Immediately"
- "To Schedule Data Backup"
- "To Schedule Incremental Data Backup"

Note

When backing up backend databases at the file system level rather than using the **backup** command, be aware that you may need to stop the server before running the backup procedure. A database backend performs internal cleanup operations that change the database log files even when there are no pending client or replication operations. An ongoing file system backup operation may therefore record database log files that are not in sync with each other. Such inconsistencies make it impossible for the backend database to recover after the database log files are restored. When the directory server is online during file system backup, successful recovery after restore can only be guaranteed if the backup operation took a fully atomic snapshot, capturing the state of all files at exactly the same time. If the recursive file system copy takes a true snapshot, you may perform the backup with the DS server online. Otherwise, if the file system copy is not a fully atomic snapshot, then you *must stop the DS server before performing the backup operation*.

To Back Up Data Immediately

To perform online backup, you start backup as a task by connecting to the administrative port and authenticating as a user with the `backend-backup` privilege, and also setting a start time for the task by using the `--start` option.

To perform offline backup when a directory server is stopped, you run the **backup** command, described in "*backup — back up directory data*" in the *Reference*, without connecting to the server, authenticating, or requesting a backup task.

- Use one of the following alternatives:
  - Back up only the database for Example.com, where the data is stored in the backend named `dsEvaluation`.

    The following example requests an online backup task that starts immediately, backing up only the `dsEvaluation` backend:

    ```
    $ backup \
     --hostname opendj.example.com \
     --port 4444 \
     --bindDN "cn=Directory Manager" \
     --bindPassword password \
     --backendID dsEvaluation \
     --backupDirectory /path/to/opendj/bak/dsEvaluation \
     --start 0 \
     --trustAll
    ```

  - Stop the server to back up Example.com data offline.

    The following example stops the server, runs offline backup, and starts the server after backup has completed:

    ```
    $ stop-ds --quiet
    $ backup --offline --backendID dsEvaluation --backupDirectory /path/to/opendj/bak/dsEvaluation
    $ start-ds --quiet
    ```

  - Back up all user data on the server.

    The following example requests an online backup task that starts immediately, backing up all backends:

    ```
    $ backup \
     --hostname opendj.example.com \
     --port 4444 \
     --bindDN "cn=Directory Manager" \
     --bindPassword password \
     --backUpAll \
     --backupDirectory /path/to/opendj/bak \
     --start 0 \
     --trustAll
    ```

To Schedule Data Backup

You can schedule online data backup using **crontab** format.

- Back up all user data every night at 2 AM, and notify diradmin@example.com when finished, or on error:

```
$ backup \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backUpAll \
 --backupDirectory /path/to/opendj/bak \
 --recurringTask "00 02 * * *" \
 --completionNotify diradmin@example.com \
 --errorNotify diradmin@example.com \
 --trustAll
```

To Schedule Incremental Data Backup

You can schedule an incremental backup by using the `--incremental` option. If you do not set the `--incrementalBaseID` option, then the server increments based on the last backup taken.

- Back up `dsEvaluation` backend data incrementally every night at 3 AM. Notify `diradmin@example.com` when finished, or on error:

```
$ backup \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backupDirectory /path/to/opendj/bak/dsEvaluation \
 --backendID dsEvaluation \
 --incremental \
 --recurringTask "00 03 * * *" \
 --completionNotify diradmin@example.com \
 --errorNotify diradmin@example.com \
 --trustAll
```

Restoring Directory Data From Backup

When you restore data, the procedure to follow depends on whether the directory server is replicated.

To Restore a Standalone Server

To restore directory data when the server is online, you start a restore task by connecting to the administrative port and authenticating as a user with the `backend-restore` privilege. You set a start time for the task with the `--start` option.

To restore data when the server is offline, you run the **restore** command, described in "*restore — restore directory data backups*" in the *Reference*, without connecting to the server, authenticating, or requesting a restore task.

- Use one of the following alternatives:
  - Stop the server to restore data for Example.com.

    The following example stops the server, restores data offline from one of the available backups, and starts the server after the restore is complete:

    ```
    $ stop-ds --quiet
    $ restore --offline --backupDirectory /path/to/opendj/bak/dsEvaluation --listBackups
    Backup ID:          <BACKUP_ID1>
    Backup Date:        <DATESTAMP1>
    Is Incremental:     false
    Is Compressed:      false
    Is Encrypted:       false
    Has Unsigned Hash:  false
    Has Signed Hash:    false
    Dependent Upon:     none

    Backup ID:          <BACKUP_ID2>
    Backup Date:        <DATESTAMP2>
    Is Incremental:     false
    Is Compressed:      false
    Is Encrypted:       false
    Has Unsigned Hash:  false
    Has Signed Hash:    false
    Dependent Upon:     none
    $ restore --offline --backupDirectory /path/to/opendj/bak/dsEvaluation --backupID ${BACKUP_ID1}
    $ start-ds --quiet
    ```

  - Schedule the restore as a task to begin immediately.

    The following example requests an online restore task, scheduled to start immediately:

```
$ restore \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backupDirectory /path/to/opendj/bak/dsEvaluation \
 --backupID ${BACKUP_ID1} \
 --start 0 \
 --trustAll
```

To Restore a Replica

After you restore a replica from backup, replication brings the replica up to date with changes that happened after you created the backup. In order to bring the replica up to date, replication must apply changes that happened after the backup was made. Replication uses internal change log records to determine what changes to apply.

Internal change log records are not kept forever, though. Replication is configured to purge the change log of old changes, preventing the log from growing indefinitely. Yet, for replication to determine what changes to apply to a restored replica, it must find change log records dating back at least to the last change in the backup. In other words, replication can bring the restored replica up to date *as long as the change log records used to determine which changes to apply have not been purged*.

Therefore, when you restore a replicated server from backup, make sure the backup you use is newer than the last purge of the replication change log (default: 3 days). If all your backups are older than the replication purge delay, do not restore from a backup, but instead initialize a new replica as described in "Initializing Replicas".

1. (Optional) When restoring data from encrypted backup, configure replication between the new replica server and a server from the existing topology as described in "Configuring Replication".

   If the backup is not encrypted, you can skip this step.

   This step initiates the server's key distribution capability, which makes it possible for the replica to obtain secret keys for decrypting backup data from existing replicas. Without the secret key for decryption, the new server cannot read the encrypted backup to restore.

   Important

   After a disaster leading to the loss of all servers in the replication topology, you must first restore a server from file system backup as described in "Backing Up Directory Data".

   When the restored server is running, configure replication between the new replica server and the restored server.

   It is not necessary to initialize replication in this step, as you will restore the data in the next step.

2. Restore the server database from the backup archive that you are sure is newer than the last purge of the replication change log.

   For examples of restoring from backup, see "To Restore a Standalone Server".

Backing Up and Restoring Configuration Files

When you back up directory data using the DS tools, you do not back up server configuration files. Server configuration files are found in the `/path/to/opendj/config/` directory.

Snapshots of the main server configuration file, `config.ldif`, are found in the `var` directory in the server instance directory. You can use these snapshots to recover from misconfiguration performed with the **dsconfig** command. The snapshots do not, however, include any other files in the `config/` directory.

Stop the server during backup and restore of configuration files as described in the following procedures:

- "To Back Up Server Configuration Files"
- "To Restore Server Configuration Files"

To Back Up Server Configuration Files

Follow these steps to back up server configuration files safely:

1. Stop the DS server:

   ```
   $ stop-ds --quiet
   ```

2. Back up the server configuration files:

   ```
   $ cd /path/to/opendj
   $ zip -r backup-config-$(date +%s).zip config
   ```

3. Start the DS server:

   ```
   $ start-ds --quiet
   ```

To Restore Server Configuration Files

Follow these steps to restore server configuration files backed up according to "To Back Up Server Configuration Files":

1. Stop the DS server:

```
$ stop-ds --quiet
```

2. Restore the server configuration files from backup, overwriting existing files:

```
$ cd /path/to/opendj
$ unzip -o backup-config-<date>.zip
```

3. Start the server:

```
$ start-ds --quiet
```

Chapter 10

Configuring Password Policy

This chapter covers password policy including examples for common use cases. In this chapter you will learn to:

- Decide what type of password policy is needed
- Discover which password policy applies for a given user
- Configure server-based and subentry-based password policies
- Assign password policies to users and to groups
- Configure automated password generation, password storage schemes, and validation of new passwords to reject invalid passwords before they are set

If you want to synchronize password policy across your organization and your applications go to the directory for authentication, then the directory can be a good place to enforce your password policy uniformly. Even if you do not depend on the directory for all your password policy, you no doubt still want to consider directory password policy if only to choose the appropriate password storage scheme.

About DS Password Policies

DS password policies govern not only passwords, but also account lockout, and how DS servers provide notification about account status.

DS servers support password policies as part of the server configuration, and subentry password policies as part of the (replicated) user data.

Server-Based Password Policies

You manage server-based password policies in the DS server configuration by using the **dsconfig** command. As they are part of the server configuration, such password policies are not replicated. You must instead apply password policy configuration updates to each replica in your deployment.

By default, DS directory servers include two password policy configurations, one default for all users, and another for the default directory superuser, `cn=Directory Manager`. You can see all the default password policy settings by using the **dsconfig** command as follows:

```
$ dsconfig \
 get-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --advanced \
 --trustAll \
 --no-prompt
Property                                  : Value(s)
------------------------------------------:-----------------------------------
account-status-notification-handler       : -
allow-expired-password-changes            : false
allow-multiple-password-values            : false
allow-pre-encoded-passwords               : false
allow-user-password-changes               : true
default-password-storage-scheme           : Salted SHA-512
deprecated-password-storage-scheme        : -
expire-passwords-without-warning          : false
force-change-on-add                       : false
force-change-on-reset                     : false
grace-login-count                         : 0
idle-lockout-interval                     : 0 s
java-class                                : org.opends.server.core.PasswordPoli
                                          : cyFactory
last-login-time-attribute                 : -
last-login-time-format                    : -
lockout-duration                          : 0 s
lockout-failure-count                     : 0
lockout-failure-expiration-interval       : 0 s
max-password-age                          : 0 s
max-password-reset-age                    : 0 s
min-password-age                          : 0 s
password-attribute                        : userPassword
password-change-requires-current-password : false
password-expiration-warning-interval      : 5 d
password-generator                        : Random Password Generator
password-history-count                    : 0
password-history-duration                 : 0 s
password-validator                        : -
previous-last-login-time-format           : -
require-change-by-time                    : -
require-secure-authentication             : false
require-secure-password-changes           : false
skip-validation-for-administrators        : false
state-update-failure-policy               : reactive
```

For detailed descriptions of each property, see "Password Policy" in the *Configuration Reference*.

Notice that many capabilities are not set by default: no lockout, no password expiration, no multiple passwords, no password validator to check that passwords contain the appropriate mix of characters. This means that if you decide to use the directory to enforce password policy, you must configure at least the default password policy to meet your needs.

A few basic protections are configured by default. When you import LDIF with `userPassword` values, DS directory servers applies a one-way hash to the values before storing them. When a user provides a password value during a bind, for example, the server hashes the value provided and compares it with the stored value. This prevents even the directory manager from recovering the plain text value of a user's password:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 userpassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userpassword: {SSHA512}<hash>
```

In addition, users can change their passwords provided that you have granted them access to do so. DS directory servers use the `userPassword` attribute to store passwords by default, rather than the `authPassword` attribute, which is designed to store passwords hashed by the client application.

Subentry-Based Password Policies

You manage subentry password policies by adding the subentries alongside the user data. DS directory servers can therefore replicate subentry password policies. The advantages are that you only need to add them once, and that administrators of user data can edit subentry password policies without having directory administrator access.

Subentry password policies support the Internet-Draft Password Policy for LDAP Directories (version 09). A subentry password policy effectively overrides settings in the default password policy defined in the DS server configuration. Settings not supported or not included in the subentry password policy are inherited from the default password policy.

"Subentry Policy Attributes vs. Server Properties" lists Internet-Draft password policy attributes that override the default password policy when you set them in the subentry:

Subentry Policy Attributes vs. Server Properties

Internet-Draft Policy Attribute	Overrides This Server Policy Property
`pwdAllowUserChange`	`allow-user-password-changes`
`pwdMustChange`	`force-change-on-reset`
`pwdGraceAuthNLimit`	`grace-login-count`
`pwdLockoutDuration`	`lockout-duration`
`pwdMaxFailure`	`lockout-failure-count`
`pwdFailureCountInterval`	`lockout-failure-expiration-interval`
`pwdMaxAge`	`max-password-age`

Internet-Draft Policy Attribute	Overrides This Server Policy Property
`pwdMinAge`	`min-password-age`
`pwdAttribute`	`password-attribute`
`pwdSafeModify`	`password-change-requires-current-password`
`pwdExpireWarning`	`password-expiration-warning-interval`
`pwdInHistory`	`password-history-count`

The following Internet-Draft password policy attributes are not taken into account by DS servers:

- `pwdCheckQuality`, because DS servers have password validators. Set password validators to use in the default password policy, for example.
- `pwdMinLength`, because this is handled by the length-based password validator. Configure this validator as part of the default password policy, for example.
- `pwdLockout`, because DS servers can deduce whether lockout is configured based on the values of other lockout-related password policy attributes.

Values of the following properties are inherited from the default password policy for Internet-Draft based password policies:

- `account-status-notification-handlers`
- `allow-expired-password-changes`
- `allow-multiple-password-values`
- `allow-pre-encoded-passwords`
- `default-password-storage-schemes`
- `deprecated-password-storage-schemes`
- `expire-passwords-without-warning`
- `force-change-on-add`
- `idle-lockout-interval`
- `last-login-time-attribute`
- `last-login-time-format`
- `max-password-reset-age`
- `password-generator`
- `password-history-duration`

- `password-validators`
- `previous-last-login-time-formats`
- `require-change-by-time`
- `require-secure-authentication`
- `require-secure-password-changes`
- `skip-validation-for-administrators`
- `state-update-failure-policy`

If you would rather specify password validators for your policy, you can configure password validators for a subentry password policy by adding the auxiliary object class `pwdValidatorPolicy` and setting the multi-valued attribute, `ds-cfg-password-validator`, to the DNs of the password validator configuration entries.

The following example shows a subentry password policy that references two password validator configuration entries. The Character Set password validator determines whether a proposed password is acceptable by checking whether it contains a sufficient number of characters from one or more user-defined character sets and ranges. The length-based password validator determines whether a proposed password is acceptable based on whether the number of characters it contains falls within an acceptable range of values. Both are enabled in the default DS server configuration:

```
dn: cn=Subentry Password Policy with Validators,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: pwdPolicy
objectClass: pwdValidatorPolicy
cn: Subentry Password Policy with Validators
pwdAttribute: userPassword
pwdLockout: TRUE
pwdMaxFailure: 3
pwdFailureCountInterval: 300
pwdLockoutDuration: 300
pwdAllowUserChange: TRUE
pwdSafeModify: TRUE
ds-cfg-password-validator: cn=Character Set,cn=Password Validators,cn=config
ds-cfg-password-validator: cn=Length-Based Password Validator,
 cn=Password Validators,cn=config
subtreeSpecification: {base "ou=people", specificationFilter
  "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)" }
```

If a referenced password validator cannot be found, then the DS server logs an error message when the password policy is invoked. This can occur, for example, when a subentry password policy is replicated to a directory server where the password validator is not (yet) configured. In that case, when a user attempts to change their password, the server fails to find the referenced password validator.

See also "To Create a Subentry-Based Password Policy".

Which Password Policy Applies

The password policy that applies to a user is identified by the operational attribute, `pwdPolicySubentry`. The default global access control instructions prevent this operational attribute from being visible to normal users. The following example gives access to administrators:

```
$ cat manage-pwp.ldif
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "pwdPolicySubentry||ds-pwp-password-policy-dn")
 (version 3.0;acl "Allow Administrators to manage user's password policy";
 allow (all) (groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com");)

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 manage-pwp.ldif
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 pwdPolicySubentry
dn: uid=bjensen,ou=People,dc=example,dc=com
pwdPolicySubentry: cn=Default Password Policy,cn=Password Policies,cn=config
```

Configuring Password Policies

You configure server-based password policies by using the **dsconfig** command. Notice that server-based password policies are part of the server configuration, and therefore not replicated. Alternatively, you can configure a subset of password policy features by using subentry-based password policies that are stored with the replicated server data.

This section covers both server-based and subentry-based password policies. It includes the following procedures:

- "To Adjust the Default Password Policy"
- "To Configure the Default Policy to Meet NIST Requirements"
- "To Create a Server-Based Password Policy"
- "To Create a Subentry-Based Password Policy"

To Adjust the Default Password Policy

You can reconfigure the default password policy, for example, to check that passwords do not contain complete attribute values, and to prevent password reuse. The default policy is a server-based password policy.

1. Apply the changes to the default password policy:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set password-history-count:7 \
 --set password-validator:Attribute\ Value \
 --trustAll \
 --no-prompt
```

2. Check your work:

```
$ dsconfig \
 get-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --trustAll
Property                                 : Value(s)
-----------------------------------------:-------------------------
account-status-notification-handler      : -
allow-expired-password-changes           : false
allow-user-password-changes              : true
default-password-storage-scheme          : Salted SHA-512
deprecated-password-storage-scheme       : -
expire-passwords-without-warning         : false
force-change-on-add                      : false
force-change-on-reset                    : false
grace-login-count                        : 0
idle-lockout-interval                    : 0 s
last-login-time-attribute                : -
last-login-time-format                   : -
lockout-duration                         : 0 s
lockout-failure-count                    : 0
lockout-failure-expiration-interval      : 0 s
max-password-age                         : 0 s
max-password-reset-age                   : 0 s
min-password-age                         : 0 s
password-attribute                       : userPassword
password-change-requires-current-password : false
password-expiration-warning-interval     : 5 d
password-generator                       : Random Password Generator
password-history-count                   : 7
password-history-duration                : 0 s
password-validator                       : Attribute Value
```

```
previous-last-login-time-format           : -
require-change-by-time                    : -
require-secure-authentication             : false
require-secure-password-changes           : false
```

3. Test changes to the default password policy.

   For example, the following tests demonstrate how the attribute value password validator works. The attribute value password validator rejects a new password when the password is contained in attribute values on the user's entry.

   By default, the attribute value password validator checks all attributes, checks whether portions of the password string match attribute values, where the portions are strings of length 5, and checks the reverse of the password as well:

```
$ dsconfig \
 get-password-validator-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --validator-name Attribute\ Value \
 --trustAll \
 --no-prompt
Property               : Value(s)
-----------------------:-------------------------------------------------
check-substrings       : true
enabled                : true
match-attribute        : All attributes in the user entry will be checked.
min-substring-length   : 5
test-reversed-password : true
```

   Consider the attributes present on Babs Jensen's entry:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)"
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
mail: bjensen@example.com
roomNumber: 0209
preferredLanguage: en, ko;q=0.8
manager: uid=trigden, ou=People, dc=example,dc=com
ou: Product Development
ou: People
givenName: Barbara
telephoneNumber: +1 408 555 1862
sn: Jensen
cn: Barbara Jensen
cn: Babs Jensen
homeDirectory: /home/bjensen
facsimileTelephoneNumber: +1 408 555 1992
gidNumber: 1000
uidNumber: 1076
description: Original description
uid: bjensen
l: San Francisco
departmentNumber: 3001
street: 201 Mission Street Suite 2900
```

Using the attribute value password validator, passwords like `bjensen12` and `babsjensenspwd` are not valid because substrings of the password match complete attribute values:

```
$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword hifalutin \
 --newPassword bjensen12
The LDAP password modify operation failed: 19 (Constraint Violation)
Additional Information:  The provided new password failed the validation
checks defined in the server: The provided password was found in another
attribute in the user entry

$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword hifalutin \
 --newPassword babsjensenspwd
The LDAP password modify operation failed: 19 (Constraint Violation)
Additional Information:  The provided new password failed the validation
checks defined in the server: The provided password was found in another
attribute in the user entry
```

The attribute value password validator does not check, however, whether the password contains substrings of attribute values:

```
$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword hifalutin \
 --newPassword babsp4ssw0rd
The LDAP password modify operation was successful

$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword babsp4ssw0rd \
 --newPassword example.com
The LDAP password modify operation was successful
```

To avoid the problem of the latter example, you could use a dictionary password validator where the dictionary includes `example.com`. For an example using a dictionary password validator, see "To Configure the Default Policy to Meet NIST Requirements".

To Configure the Default Policy to Meet NIST Requirements

As described in "To Adjust the Default Password Policy", the default policy is a server-based password policy.

You can change the default password policy to use following rules that are inspired by NIST 800-63 requirements (as of September 2016):

- Use a strong password storage scheme.

  The example in this procedure uses PBKDF2, which requires more processing time than Salted SHA-512 (the default).

- Enforce a minimum password length of 8 characters.

- Check for matches in a dictionary of compromised passwords.

  Use a file in the same format as `config/wordlist.txt`, where each line contains a common password. Lists of common passwords can be found online.

- Do not use composition rules for password validation.

  In other words, do not require a mix of special characters, upper and lower case letters, numbers, or other composition rules.

- Do not enforce arbitrary password changes.

  In other words, do not set a maximum password age.

Follow these steps to make this the default password policy:

1. Create a password validator for a minimum length of 8 characters:

```
$ dsconfig \
 create-password-validator \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --validator-name "At least 8 characters" \
 --type length-based \
 --set enabled:true\
 --set min-password-length:8 \
 --trustAll \
 --no-prompt
```

2. Create a password validator for checking common compromised passwords:

```
# Obtain a copy of a dictionary list of common passwords:
$ cp 10k_most_common.txt /path/to/opendj/config/
$ dsconfig \
 create-password-validator \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --validator-name "Common passwords" \
 --type dictionary \
 --set enabled:true \
 --set dictionary-file:config/10k_most_common.txt \
 --trustAll \
 --no-prompt
```

3. Configure the password policy as the default:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set default-password-storage-scheme:PBKDF2 \
 --set password-validator:"At least 8 characters" \
 --set password-validator:"Common passwords" \
 --trustAll \
 --no-prompt
```

4. Check that the default password policy works appropriately.

   The following example shows a rejected password modification:

```
$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword hifalutin \
 --newPassword secret12
The LDAP password modify operation failed: 19 (Constraint Violation)
Additional Information:  The provided new password failed the validation
checks defined in the server: The provided password was found in another
attribute in the user entry
```

The following example shows an accepted password modification:

```
$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword hifalutin \
 --newPassword aET1OjQeVJECSMgxDPs3U6In
The LDAP password modify operation was successful
```

To Create a Server-Based Password Policy

You can add a password policy, for example, for new users who have not yet used their credentials to bind.

1. Create the new password policy:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "New Account Password Policy" \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set force-change-on-add:true \
 --set password-attribute:userPassword \
 --type password-policy \
 --trustAll \
 --no-prompt
```

2. Check your work:

```
$ dsconfig \
 get-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "New Account Password Policy" \
```

```
 --trustAll
Property                                  : Value(s)
------------------------------------------:---------------
account-status-notification-handler       : -
allow-expired-password-changes            : false
allow-user-password-changes               : true
default-password-storage-scheme           : Salted SHA-512
deprecated-password-storage-scheme        : -
expire-passwords-without-warning          : false
force-change-on-add                       : true
force-change-on-reset                     : false
grace-login-count                         : 0
idle-lockout-interval                     : 0 s
last-login-time-attribute                 : -
last-login-time-format                    : -
lockout-duration                          : 0 s
lockout-failure-count                     : 0
lockout-failure-expiration-interval       : 0 s
max-password-age                          : 0 s
max-password-reset-age                    : 0 s
min-password-age                          : 0 s
password-attribute                        : userPassword
password-change-requires-current-password : false
password-expiration-warning-interval      : 5 d
password-generator                        : -
password-history-count                    : 0
password-history-duration                 : 0 s
password-validator                        : -
previous-last-login-time-format           : -
require-change-by-time                    : -
require-secure-authentication             : false
require-secure-password-changes           : false
```

If you use a password policy like this, then you will want to change the user's policy again when the new user successfully updates the password. For instructions on assigning a server-based password policy, see "To Assign a Password Policy to a User".

To Create a Subentry-Based Password Policy

You can add a subentry to configure a password policy that applies to Directory Administrators.

1. Create the entry that specifies the password policy:

```
$ cat subentry-password-policy.ldif
dn: cn=Subentry Password Policy with Validators,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: pwdPolicy
objectClass: pwdValidatorPolicy
cn: Subentry Password Policy with Validators
pwdAttribute: userPassword
pwdLockout: TRUE
pwdMaxFailure: 3
pwdFailureCountInterval: 300
pwdLockoutDuration: 300
pwdAllowUserChange: TRUE
pwdSafeModify: TRUE
ds-cfg-password-validator: cn=Character Set,cn=Password Validators,cn=config
ds-cfg-password-validator: cn=Length-Based Password Validator,
 cn=Password Validators,cn=config
subtreeSpecification: {base "ou=people", specificationFilter
  "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)" }
```

The `base` entry identifies the branch that holds administrator entries. For details on how subentries apply, see "Understanding Subentry Scope".

2. Add the policy to the directory:

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

3. Check that the policy applies as specified.

   In this example, the policy should apply to a Directory Administrator, while a normal user has the default password policy. Here, Kirsten Vaughan is a member of the Directory Administrators group, and Babs Jensen is not a member:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 pwdPolicySubentry
dn: uid=kvaughan,ou=People,dc=example,dc=com
pwdPolicySubentry: cn=Subentry Password Policy with Validators,dc=example,dc=com

$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 pwdPolicySubentry
dn: uid=bjensen,ou=People,dc=example,dc=com
pwdPolicySubentry: cn=Default Password Policy,cn=Password Policies,cn=config
```

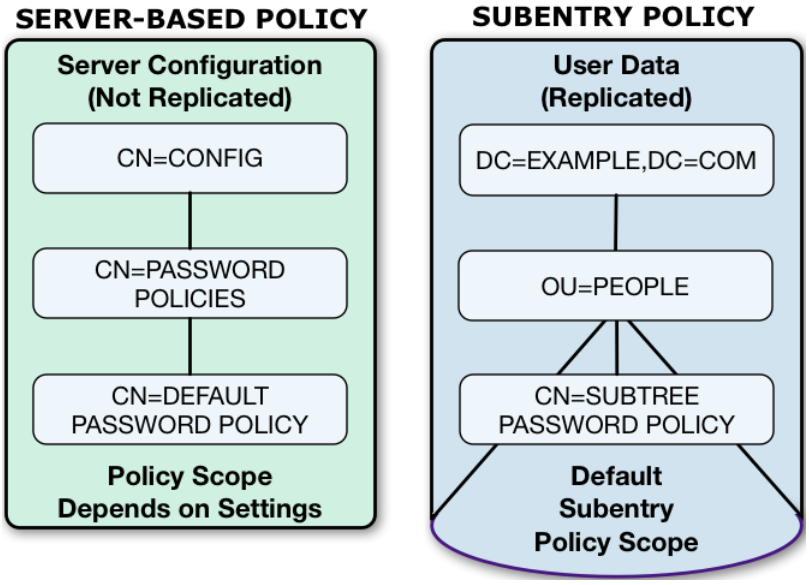
Assigning Password Policies

You assign subentry-based password policies for a subtree of the DIT by adding the policy to an LDAP subentry whose immediate superior is the root of the subtree. In other words, you can add the subentry-based password policy under `ou=People,dc=example,dc=com` to have it apply to all entries under `ou=People,dc=example,dc=com`. You can also use the capabilities of LDAP subentries to refine the scope of application.

You assign server-based password policies by using the `ds-pwp-password-policy-dn` attribute.

"Server-Based and Subentry Password Policies" compares the types of password policy.

Server-Based and Subentry Password Policies



To Assign a Password Policy to a User

1. Give administrators the right to manage users' password policies:

```
$ cat manage-pwp.ldif
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "pwdPolicySubentry||ds-pwp-password-policy-dn")
 (version 3.0;acl "Allow Administrators to manage user's password policy";
 allow (all) (groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com");)

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

Notice here that the directory superuser, `cn=Directory Manager`, assigns privileges to Kirsten Vaughan. Any administrator with the `privilege-change` privilege can assign other privileges. Assuming the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both

user data ACIs and global ACIs. For this reason, avoid assigning the `privilege-change` privilege to normal administrator users.

2. Update the user's `ds-pwp-password-policy-dn` attribute:

```
$ cat newuser.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
uid: newuser
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: New User
sn: User
ou: People
mail: newuser@example.com
userPassword: changeme
ds-pwp-password-policy-dn: cn=New Account Password Policy,cn=Password Policies,cn=config

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

3. Check your work:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=newuser)" \
 pwdPolicySubentry
dn: uid=newuser,ou=People,dc=example,dc=com
pwdPolicySubentry: cn=New Account Password Policy,cn=Password Policies,cn=config
```

To Assign a Password Policy to a Group

You can use a collective attribute to assign a password policy. Collective attributes provide a standard mechanism for defining attributes that appear on all the entries in a subtree. For details, see "Collective Attributes" in the *Developer's Guide*:

1. Give an administrator the privilege to write subentries, such as those used for setting collective attributes:

```
$ cat subentry-write.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: subentry-write

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

Notice here that the directory superuser, `cn=Directory Manager`, assigns privileges to Kirsten Vaughan. Any administrator with the `privilege-change` privilege can assign other privileges. Assuming the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, avoid assigning the `privilege-change` privilege to normal administrator users.

2. Create a subentry defining the collective attribute that sets the `ds-pwp-password-policy-dn` attribute for group members' entries:

```
$ cat pwp-coll.ldif
dn: cn=Password Policy for Dir Admins,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Password Policy for Dir Admins
ds-pwp-password-policy-dn;collective: cn=Root Password Policy,cn=Password Policies,cn=config
subtreeSpecification: { base "ou=People", specificationFilter
  "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)"}

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

3. Check your work:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 pwdPolicySubentry
dn: uid=kvaughan,ou=People,dc=example,dc=com
pwdPolicySubentry: cn=Root Password Policy,cn=Password Policies,cn=config
```

To Assign Password Policy for an Entire Branch

A password policy subentry password policy to assign the policy to the entries under a base DN:

1. Give an administrator the privilege to write subentries, such as those used for setting password policies:

```
$ cat subentry-write.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: subentry-write

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

   Notice here that the directory superuser, `cn=Directory Manager`, assigns privileges to Kirsten Vaughan. Any administrator with the `privilege-change` privilege can assign other privileges. Assuming the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, avoid assigning the `privilege-change` privilege to normal administrator users.

2. Create a password policy with a `subtreeSpecification` to assign the policy to all entries under a base DN.

   The following example creates a password policy for entries under `ou=People,dc=example,dc=com`:

```
$ cat people-pwp.ldif
dn: cn=People Password Policy,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: pwdPolicy
cn: People Password Policy
pwdAttribute: userPassword
pwdLockout: TRUE
pwdMaxFailure: 3
pwdFailureCountInterval: 300
pwdLockoutDuration: 300
pwdAllowUserChange: TRUE
pwdSafeModify: TRUE
subtreeSpecification: { base "ou=people" }

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

Notice the subtree specification used to assign the policy, `{ base "ou=people" }`. You can relax the subtree specification value to `{}` to apply the password policy to all sibling entries (all entries under `dc=example,dc=com`), or further restrict the subtree specification by adding a `specificationFilter`. See "Understanding Subentry Scope" for more information.

3. Check your work:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=alutz)" \
 pwdPolicySubentry
dn: uid=alutz,ou=People,dc=example,dc=com
pwdPolicySubentry: cn=People Password Policy,dc=example,dc=com
```

If everything is correctly configured, then the password policy should be assigned to users whose entries are under `ou=People,dc=example,dc=com`.

Understanding Subentry Scope

LDAP subentries reside with the user data and so are replicated. Subentries hold operational data. They are not visible in search results unless explicitly requested. This section describes how a subentry's `subtreeSpecification` attribute defines the scope of the subtree that the subentry applies to.

An LDAP subentry's subtree specification identifies a subset of entries in a branch of the DIT. The subentry scope is these entries. In other words, these are the entries that the subentry affects.

The attribute value for a `subtreeSpecification` optionally includes the following parameters:

`base`

Indicates the entry, *relative to the subentry's parent*, at the base of the subtree.

By default, the base is the subentry's parent.

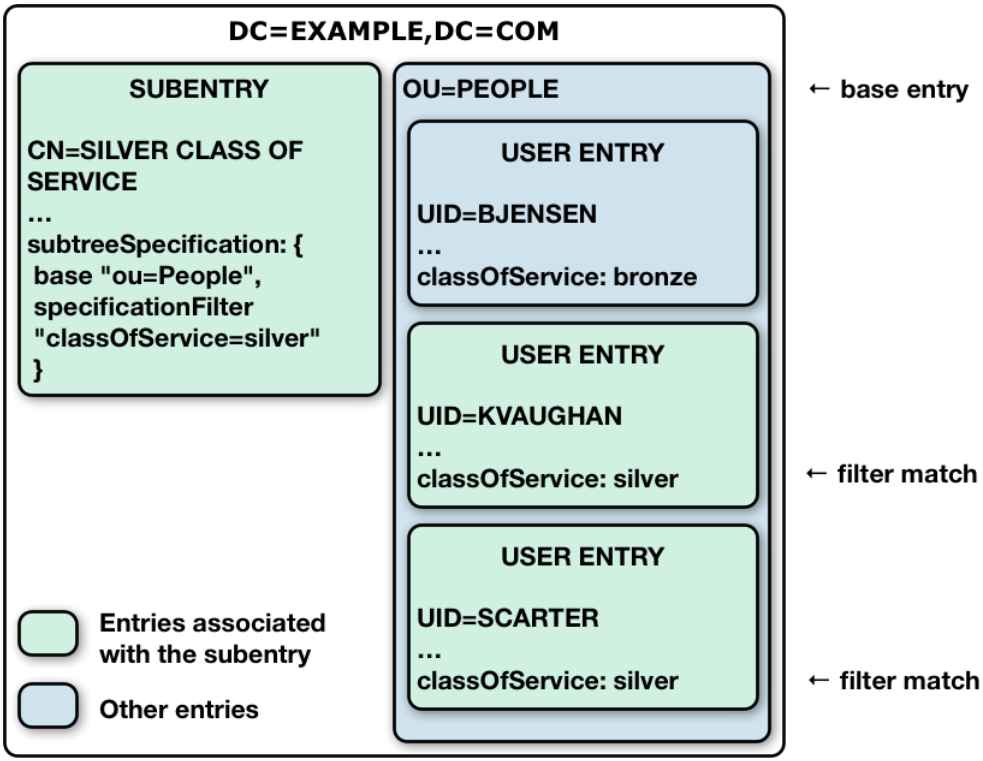
`specificationFilter`

Indicates an LDAP filter. Entries matching the filter are in scope.

By default, all entries under the base entry are in scope.

"Subtree Associated With a Subentry" illustrates these characteristics for an example collective attribute subentry.

Subtree Associated With a Subentry



Notice that the base of `ou=People` on the subentry `cn=Silver Class of Service,dc=example,dc=com` indicates that the base entry is `ou=People,dc=example,dc=com`.

The filter `"(classOfService=silver)"` means that Kirsten Vaughan and Sam Carter's entries are in scope. Babs Jensen's entry, with `classOfService: bronze` does not match and is therefore not in scope. The `ou=People` organizational unit entry does not have a `classOfService` attribute, and so is not in scope, either.

Configuring Password Generation

Password generators are used by DS servers during the LDAP Password Modify extended operation to construct a new password for the user. In other words, a directory administrator resetting a user's password can have a server generate the new password by using the **ldappasswordmodify** command, described in "*ldappasswordmodify — perform LDAP password modifications*" in the *Reference*:

```
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --authzID "u:bjensen"
The LDAP password modify operation was successful
Generated Password:  <random>
```

The default password policy shown in "To Adjust the Default Password Policy" uses the Random Password Generator, described in "Random Password Generator" in the *Configuration Reference*:

```
$ dsconfig \
 get-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --property password-generator \
 --trustAll \
 --no-prompt
Property           : Value(s)
-------------------:-------------------------
password-generator : Random Password Generator
$ dsconfig \
 get-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --property password-generator \
 --trustAll \
 --no-prompt
Property           : Value(s)
-------------------:-------------------------
password-generator : Random Password Generator
```

Notice that the default configuration for the Random Password Generator defines two `password-character-set` values, and then uses those definitions in the `password-format` so that generated passwords have eight characters: three from the `alpha` set, followed by two from the `numeric` set, followed by three from the `alpha` set. The `password-character-set` name must be ASCII.

To set the password generator that the DS server employs when constructing a new password for a user, set the `password-generator` property for the password policy that applies to the user.

The following example does not change the password policy, but instead changes the Random Password Generator configuration, and then demonstrates a password being generated upon reset:

```
$ dsconfig \
 set-password-generator-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --generator-name "Random Password Generator" \
 --remove password-character-set:alpha:abcdefghijklmnopqrstuvwxyz \
 --add password-character-set:alpha:ABCDEFGHIJKLMNOPQRSTUVWabcdefghijklmnopqrstuvwxyz \
 --add password-character-set:"punct:,./\`!@#\$%^&*:\;[]\"\'\(\)+=-_~\\" \
 --set password-format:alpha:3,punct:1,numeric:2,punct:2,numeric:3,alpha:3,punct:2 \
 --trustAll \
 --no-prompt
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --authzID "u:bjensen"
The LDAP password modify operation was successful
Generated Password:  <random>
```

If you also set up a password validator in the password policy as shown in "To Adjust the Default Password Policy" and further described in "Configuring Password Validation", make sure the generated passwords are acceptable to the validator.

Configuring Password Storage

Password storage schemes, described in "Password Storage Scheme" in the *Configuration Reference*, encode new passwords and store the encoded version. When a client application authenticates with the password, the server encodes the cleartext password using the configured storage scheme, and checks whether the result matches the encoded value stored by the server. If the encoded version is appropriately secure, it is difficult to guess the cleartext password from its encoded value.

DS servers offer a variety of reversible and one-way password storage schemes. With a reversible encryption scheme, an attacker who gains access to the server can recover the cleartext passwords. With a one-way hash storage scheme, the attacker who gains access to the server must still crack the password by brute force, encoding passwords over and over to generate guesses until a match is found. If you have a choice, use a one-way password storage scheme.

Some one-way hash functions are not designed specifically for password storage, but also for use in message authentication and digital signatures. Such functions, like those defined in the Secure Hash Algorithm (SHA-1 and SHA-2) standards, are designed for high performance. Because they are fast, they allow the server to perform authentication at high throughput with low response times. However, high-performance algorithms also help attackers use brute force techniques. One estimate in 2017 is that a single GPU can calculate over one billion SHA-512 hashes per second.

Warning

Some one-way hash functions are designed to be computationally *expensive*. Such functions, like PBKDF2 and Bcrypt, are designed to be relatively slow even on modern hardware. This makes them generally less susceptible to brute force attacks. *However*, computationally expensive functions reduce authentication throughput and increase response times. With the default number of iterations, the GPU mentioned above might only calculate 100,000 PBKDF2 hashes per second (or 0.01% of the corresponding hashes calculated with SHA-512). If you use these functions, be aware of the potentially dramatic performance impact and plan your deployment accordingly. Do not use functions like Bcrypt for any accounts that are used for frequent, short-lived connections.

Modern hardware and techniques to pre-compute attempts, such as rainbow tables, make it increasingly easy for attackers to crack passwords by brute force. Password storage schemes that use *salt* make brute force attacks more expensive. In this context, salt is a random value appended to the password before encoding. The salt is then stored with the encoded value and used when comparing an incoming password to the stored password.

Reversible password storage schemes, such as AES and Blowfish, use symmetric keys for encryption. As described in "How Secret Keys are Distributed" in the *Security Guide*, DS servers use the Crypto Manager subsystem to safely distribute symmetric keys. In summary, the Crypto Manager relies on replication to distribute symmetric keys to each replica. In order for a directory server to decrypt a password encrypted by another server, *it must be a replica of the server that encrypted the password*. For information about replication and how to use it, see "*Managing Data Replication*".

The following example lists available alternatives, further described in "Password Storage Schemes":

```
$ dsconfig \
 list-password-storage-schemes \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --trustAll
Password Storage Scheme : Type          : enabled
------------------------:---------------:--------
3DES                    : triple-des    : true
AES                     : aes           : true
Base64                  : base64        : true
Bcrypt                  : bcrypt        : true
Blowfish                : blowfish      : true
Clear                   : clear         : true
CRYPT                   : crypt         : true
MD5                     : md5           : true
PBKDF2                  : pbkdf2        : true
PKCS5S2                 : pkcs5s2       : true
RC4                     : rc4           : true
Salted MD5              : salted-md5    : true
Salted SHA-1            : salted-sha1   : true
Salted SHA-256          : salted-sha256 : true
Salted SHA-384          : salted-sha384 : true
Salted SHA-512          : salted-sha512 : true
SHA-1                   : sha1          : true
```

As shown in "To Adjust the Default Password Policy", the default password storage scheme for users is Salted SHA-512. When you add users or import user entries with `userPassword` values in cleartext, the DS server hashes them with the default password storage scheme. The default directory superuser has a different password policy, shown in "To Assign a Password Policy to a Group". The Root Password Policy uses PBKDF2 by default.

Tip

The choice of default password storage scheme for normal users can significantly impact server performance. Each time a normal user authenticates using simple bind (username/password) credentials, the directory server encodes the user's password according to the storage scheme in order to compare it with the encoded value in the user's entry.

Schemes such as Salted SHA-512 call for relatively high-performance encoding. Schemes such as PBKDF2, which are designed to make the encoding process computationally expensive, reduce the bind throughput that can be achieved on equivalent hardware.

Take this performance impact into consideration when choosing a password storage scheme. If you opt for a computationally expensive scheme such as PBKDF2, make sure the directory service has enough compute power to absorb the additional load.

Password Storage Schemes

Name	Type of Algorithm	Notes
3DES	Reversible encryption	Triple DES (Data Encryption Standard) in EDE (Encrypt Decrypt Encrypt) mode. Key size: 168 bits.
AES	Reversible encryption	Advanced Encryption Standard, successor to DES, published by the US National Institute of Standards and Technology (NIST). Key size: 128 bits.
Base64	Reversible encoding	Transfer encoding for representing binary password values in text. *Not intended as a secure storage scheme.*
Bcrypt	One-way hash	Computationally intensive hashing function, based on the Blowfish cipher. Default cost: 12 (2^12 iterations).
Blowfish	Reversible encryption	Public domain cipher designed by Bruce Schneier as a successor to DES. Key size: 128 bits.
Clear	Cleartext, no encoding	For backwards compatibility and use with certain legacy applications. *Not intended as a secure storage scheme.*
CRYPT	One-way hash	Based on the UNIX Crypt algorithm. For backwards compatibility and use with certain legacy applications. *Not intended as a secure storage scheme.* Default algorithm: `unix`.
MD5	One-way hash	Based on the MD5 algorithm defined in RFC 1321. For backwards compatibility and use with certain legacy applications. *Not intended as a secure storage scheme.*
PBKDF2	One-way hash	Computationally intensive hashing function, based on PBKDF2 algorithm defined in RFC 2898.

Name	Type of Algorithm	Notes
		Default iterations: 10000.
PKCS5S2	One-way hash	Computationally intensive hashing function, based on Atlassian's adaptation of the PBKDF2. Number of iterations: 10000.
RC4	Reversible encryption	Based on the Rivest Cipher 4 algorithm. For backwards compatibility and use with certain legacy applications. *Not intended as a secure storage scheme.* Key size: 128 bits.
Salted MD5	One-way hash	Based on MD5, with 64 bits of random salt appended to the cleartext before hashing, and then appended to the hash.
Salted SHA-1	One-way hash	Based on SHA-1, with 64 bits of random salt appended to the cleartext before hashing, and then appended to the hash.
Salted SHA-256	One-way hash	Based on the SHA-256 hash function using 32-bit words and producing 256-bit digests. SHA-256 is defined in the SHA-2 (Secure Hash Algorithm 2) standard developed by the US National Security Agency (NSA) and published by NIST. The salt is applied as for Salted SHA-1.
Salted SHA-384	One-way hash	Based on the SHA-384 hash function that effectively truncates the digest of SHA-512 to 384 bits. SHA-384 is defined in the SHA-2 (Secure Hash Algorithm 2) standard developed by the NSA and published by NIST. The salt is applied as for Salted SHA-1.
Salted SHA-512	One-way hash	Based on the SHA-512 hash function using 64-bit words and producing 512-bit digests. SHA-512 is defined in the SHA-2 (Secure Hash Algorithm 2) standard developed by the NSA and published by NIST. The salt is applied as for Salted SHA-1.
SHA-1	One-way hash	SHA-1 (Secure Hash Algorithm 1) standard developed by the NSA and published by NIST. *Not intended as a secure storage scheme.*

Password storage schemes listed in the following table have additional configuration settings.

Additional Password Storage Scheme Settings

Scheme	Setting	Description
Bcrypt	`bcrypt-cost`	The cost parameter specifies a key expansion iteration count as a power of two. A default value of 12 (2^{12} iterations) is considered in 2016 as a reasonable balance between responsiveness and security for regular users.
Crypt	`crypt-password-storage-encryption-algorithm`	Specifies the crypt algorithm to use to encrypt new passwords. The following values are supported: **`unix`** The password is encrypted with the weak Unix crypt algorithm. This is the default setting. **`md5`** The password is encrypted with the BSD MD5 algorithm and has a `$1$` prefix. **`sha256`** The password is encrypted with the SHA256 algorithm and has a `$5$` prefix. **`sha512`** The password is encrypted with the SHA512 algorithm and has a `$6$` prefix.
PBKDF2	`pbkdf2-iterations`	The number of algorithm iterations. NIST recommends at least 1000. The default is 10000.

You change the default password policy storage scheme for users by changing the applicable password policy, as shown in the following example:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set default-password-storage-scheme:pbkdf2 \
 --trustAll \
 --no-prompt
```

Notice that the change in default password storage scheme does not cause the DS server to update any stored password values. By default, the server only stores a password with the new storage scheme the next time that the password is changed.

DS servers prefix passwords with the scheme used to encode them, which means it is straightforward to see which password storage scheme is used. After the default password storage scheme is changed to PBKDF2, old user passwords remain encoded with Salted SHA-512:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

When the password is changed, the new default password storage scheme takes effect, as shown in the following example:

```
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --authzID "u:bjensen" \
 --newPassword changeit
The LDAP password modify operation was successful
$ ldapsearch \
 --port 1389 \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword changeit \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {PBKDF2}10000:<hash>
```

When you change the password storage scheme for users, realize that the user passwords must change in order for the DS server to encode them with the chosen storage scheme. If you are changing the storage scheme because the old scheme was too weak, then you no doubt want users to change their passwords anyway.

If, however, the storage scheme change is not related to vulnerability, you can use the `deprecated-password-storage-scheme` property of the password policy to have the DS server store the password in the new format after successful authentication. This makes it possible to do password migration for active users without forcing users to change their passwords:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 userPassword
dn: uid=kvaughan,ou=People,dc=example,dc=com
userPassword: {SSHA512}<hash>
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set deprecated-password-storage-scheme:"Salted SHA-1" \
 --trustAll \
 --no-prompt
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 userPassword
dn: uid=kvaughan,ou=People,dc=example,dc=com
userPassword: {PBKDF2}10000:<hash>
```

Notice that with `deprecated-password-storage-scheme` set appropriately, Kirsten Vaughan's password was hashed again after she authenticated successfully.

Configuring Password Validation

Password validators, described in "Password Validator" in the *Configuration Reference*, are responsible for determining whether a proposed password is acceptable for use. Validators can run checks like ensuring that the password meets minimum length requirements, that it has an appropriate range of characters, or that it is not in the history of recently used passwords.

The following example lists the default password validators:

```
$ dsconfig \
 list-password-validators \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --trustAll \
 --no-prompt
Password Validator                  : Type                : enabled
------------------------------------:---------------------:--------
Attribute Value                     : attribute-value     : true
Character Set                       : character-set       : true
Dictionary                          : dictionary          : false
Length-Based Password Validator     : length-based        : true
Repeated Characters                 : repeated-characters : true
Similarity-Based Password Validator : similarity-based    : true
Unique Characters                   : unique-characters   : true
```

The password policy for a user specifies the set of password validators that should be used whenever that user provides a new password. By default, no password validators are configured. You can see an example setting the Default Password Policy to use the Dictionary validator in "To Adjust the Default Password Policy". The following example shows how to set up a custom password validator and assign it to the default password policy.

The custom password validator ensures passwords meet at least three of the following four criteria. Passwords are composed of the following:

- English lowercase characters (a through z)
- English uppercase characters (A through Z)
- Base 10 digits (0 through 9)
- Non-alphabetic characters (for example, !, $, #, %)

Notice how the `character-set` values are constructed. The initial `0:` means the set is optional, whereas `1:` means the set is required:

```
$ dsconfig \
 create-password-validator \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --validator-name "Custom Character Set Password Validator" \
 --set allow-unclassified-characters:true \
 --set enabled:true \
 --set character-set:0:abcdefghijklmnopqrstuvwxyz \
 --set character-set:0:ABCDEFGHIJKLMNOPQRSTUVWXYZ \
 --set character-set:0:0123456789 \
 --set character-set:0:\!\"#\$%\&\'\(\)\*+\,-./:\;\\\<=\>\?@\[\]\^_\`\{\|\}~ \
 --set min-character-sets:3 \
```

```
 --type character-set \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set password-validator:"Custom Character Set Password Validator" \
 --trustAll \
 --no-prompt
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --authzID "u:bjensen" \
 --newPassword '!ABcd$%^'
```

In the preceding example, the character set of ASCII punctuation, `\!\"#\$%\&\'\(\)\*+\,-./:\;\\\<=\>\?@\[\]\^_\`\{\|\}~`, is hard to read because of all of the escape characters. It is equivalent to `!"#$%&'()*+,-./:;\<=>?@[]^_`{|}~`. To enter sequences sequences that are difficult to escape properly, use the **dsconfig** in interactive mode, and let it do the escaping for you. You can also use the `--commandFilePath {path}` option to save the result of your interactive session to a file for later use in scripts.

An attempt to set an invalid password fails as shown in the following example:

```
$ ldappasswordmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --authzID "u:bjensen" \
 --newPassword hifalutin
The LDAP password modify operation failed with result code 19
Error Message:  The provided new password failed the validation checks defined
in the server: The provided password did not contain characters from at least
3 of the following character sets or ranges: '0123456789',
'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz',
'!"#$%&'()*+,-./:;\<=>?@[]^_`{|}~'
```

Validation does not affect existing passwords, but only takes effect when the password is updated.

You can reference password validators from subentry password policies. See "Subentry-Based Password Policies" for an example.

Sample Password Policies

The sample password policies in this section demonstrate DS server-based password policies for several common cases:

- "Enforce Regular Password Changes"
- "Track Last Login Time"
- "Deprecate a Password Storage Scheme"
- "Lock Idle Accounts"
- "Allow Grace Log In to Change Expired Password"
- "Require Password Change on Add or Reset"

Enforce Regular Password Changes

The following commands configure a server-based password policy that sets age limits on passwords, requiring that they change periodically. It also sets the number of passwords to keep in the password history of the entry, thereby preventing users from reusing the same password on consecutive changes:

```
$ dsconfig  \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Enforce Regular Password Changes" \
 --type password-policy \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set password-attribute:userPassword \
 --set max-password-age:13w \
 --set min-password-age:4w \
 --set password-history-count:7 \
 --trustAll \
 --no-prompt
```

See also "Assigning Password Policies" for instructions on using the policy.

Track Last Login Time

The following commands configure a server-based password policy that keeps track of the last successful login.

First, set up an attribute to which the DS directory server can write a timestamp value on successful login.

The following example defines a `lastLoginTime` attribute. Notice that the attribute has the property `USAGE directoryOperation`, meaning this is an operational attribute. When checking schema compliance, the server skips operational attributes. Operational attributes can therefore be added to an entry without changing the entry's object classes. Furthermore, as described in "About Data In LDAP Directories", operational attributes hold information used by the directory itself. Operational attributes are only returned when explicitly requested, and not intended for modification by external

applications. By defining the `lastLoginTime` attribute as operational, you limit its visibility and help prevent client applications from changing its value unless specifically granted access to do so.

For additional information, also see "Search: Listing Active Accounts" in the *Developer's Guide*:

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

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

Next, create the password policy that causes the DS directory server to write the timestamp to the attribute on successful login:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Track Last Login Time" \
 --type password-policy \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set password-attribute:userPassword \
 --set last-login-time-attribute:lastLoginTime \
 --set last-login-time-format:"yyyyMMddHH'Z'" \
 --trustAll \
 --no-prompt
```

See "Assigning Password Policies" for instructions on using the policy.

Deprecate a Password Storage Scheme

The following commands configure a server-based password policy that you can use when deprecating a password storage scheme. This policy uses elements from "Enforce Regular Password Changes". The DS server applies the new password storage scheme to hash or to encrypt passwords

when a password changes, or when the user successfully binds with the correct password and the password is currently hashed with a deprecated scheme:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Deprecate a Password Storage Scheme" \
 --type password-policy \
 --set deprecated-password-storage-scheme:Crypt \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set password-attribute:userPassword \
 --set max-password-age:13w \
 --set min-password-age:4w \
 --set password-history-count:7 \
 --trustAll \
 --no-prompt
```

See also "Assigning Password Policies" for instructions on using the policy.

Lock Idle Accounts

The following commands configure a server-based password policy that locks idle accounts. This policy extends the example from "Track Last Login Time" as the DS server must track last successful login time in order to calculate how long the account has been idle. You must first add the `lastLoginTime` attribute type in order for the DS server to accept this new password policy:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Lock Idle Accounts" \
 --type password-policy \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set password-attribute:userPassword \
 --set last-login-time-attribute:lastLoginTime \
 --set last-login-time-format:"yyyyMMddHH'Z'" \
 --set idle-lockout-interval:13w \
 --trustAll \
 --no-prompt
```

See also "Assigning Password Policies", and "Configuring Account Lockout".

Allow Grace Log In to Change Expired Password

The following commands configure a server-based password policy that allows users to log in after their password has expired in order to choose a new password:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Allow Grace Login" \
 --type password-policy \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set password-attribute:userPassword \
 --set grace-login-count:2 \
 --trustAll \
 --no-prompt
```

See also "Assigning Password Policies" for instructions on using the policy.

Require Password Change on Add or Reset

The following commands configure a server-based password policy that requires new users to change their password after logging in for the first time, and also requires users to change their password after their password is reset:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Require Password Change on Add or Reset" \
 --type password-policy \
 --set default-password-storage-scheme:"Salted SHA-512" \
 --set password-attribute:userPassword \
 --set force-change-on-add:true \
 --set force-change-on-reset:true \
 --trustAll \
 --no-prompt
```

See also "Assigning Password Policies" for instructions on using the policy.

Chapter 11

Implementing Account Lockout and Notification

This chapter covers configuration of account lockout and account status notification. In this chapter you will learn to:

- Configure password policies to manage account lockout automatically
- Manage lockout with the **manage-account** command
- Set up email notification of account status

DS servers support automatic account lockout. The aim of account lockout is not to punish users who mistype their passwords, but instead to protect the directory against attacks in which the attacker attempts to guess a user password, repeatedly attempting to bind until success is achieved.

Account lockout disables a user account after a specified number of successive authentication failures. When you implement account lockout, you can opt to have the DS server unlock the account after a specified interval, or you can leave the account locked until the password is reset.

Note

You configure account lockout as part of password policy. the server locks an account after the specified number of consecutive authentication failures. Account lockout is not transactional across a replication topology. Under normal circumstances, replication propagates lockout quickly. If replication is ever delayed, an attacker with direct access to multiple replicas could try to authenticate up to the specified number of times on each replica before being locked out on all replicas.

This chapter shows you how to set up account lockout policies by using the **dsconfig** command, described in "*dsconfig — manage OpenDJ server configuration*" in the *Reference*, and how to intervene manually to lock and unlock accounts by using the **manage-account** command, described in "*manage-account — manage state of OpenDJ server accounts*" in the *Reference*.

Configuring Account Lockout

Account lockout is configured as part of password policy. This section demonstrates configuring account lockout as part of the default password policy. Users are allowed three consecutive failures before being locked out for five minutes. Failures themselves also expire after five minutes.

Change the default password policy to activate lockout using the **dsconfig** command. As the password policy is part of the server configuration, you must manually apply the changes to each replica in a replication topology:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set lockout-failure-count:3 \
 --set lockout-duration:5m \
 --set lockout-failure-expiration-interval:5m \
 --trustAll \
 --no-prompt
```

Users having the default password policy are then locked out after three failed attempts in succession:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=bjensen,ou=people,dc=example,dc=com" \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail
dn: uid=bjensen,ou=People,dc=example,dc=com
mail: bjensen@example.com
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=bjensen,ou=people,dc=example,dc=com" \
 --bindPassword fatfngrs \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail
The LDAP bind request failed: 49 (Invalid Credentials)
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=bjensen,ou=people,dc=example,dc=com" \
 --bindPassword fatfngrs \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail
The LDAP bind request failed: 49 (Invalid Credentials)
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=bjensen,ou=people,dc=example,dc=com" \
 --bindPassword fatfngrs \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail
The LDAP bind request failed: 49 (Invalid Credentials)
$ ldapsearch \
 --port 1389 \
```

```
 --bindDN "uid=bjensen,ou=people,dc=example,dc=com" \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 mail
The LDAP bind request failed: 49 (Invalid Credentials)
```

Managing Accounts Manually

This section covers disabling and enabling accounts by using the **manage-account** command. Password reset is covered in the chapter on performing LDAP operations.

To Disable an Account

1. Make sure the user running the **manage-account** command has access to perform the appropriate operations.

   Kirsten Vaughan is a member of the Directory Administrators group. For this example, she must have the `password-reset` privilege, and access to edit user attributes and operational attributes:

   ```
   $ cat manage-account-access.ldif
   dn: uid=kvaughan,ou=People,dc=example,dc=com
   changetype: modify
   add: ds-privilege-name
   ds-privilege-name: password-reset

   dn: ou=People,dc=example,dc=com
   changetype: modify
   add: aci
   aci: (target="ldap:///ou=People,dc=example,dc=com")(targetattr ="*||+")
    (version 3.0;acl "Admins can run amok"; allow(all)
     groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)

   $ ldapmodify \
    --port 1389 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    manage-account-access.ldif
   ```

   Notice here that the directory superuser, `cn=Directory Manager`, assigns privileges to Kirsten Vaughan. Any administrator with the `privilege-change` privilege can assign other privileges. Assuming the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, avoid assigning the `privilege-change` privilege to normal administrator users.

2. Set the account status to disabled with the **manage-account** command:

```
$ manage-account \
 set-account-is-disabled \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --operationValue true \
 --targetDN uid=bjensen,ou=people,dc=example,dc=com \
 --trustAll
Account Is Disabled:  true
```

To Activate a Disabled Account

- Clear the disabled status using the **manage-account** command:

```
$ manage-account \
 set-account-is-disabled \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --operationValue false \
 --targetDN uid=bjensen,ou=people,dc=example,dc=com \
 --trustAll
Account Is Disabled:  false
```

Managing Account Status Notification

DS servers can send mail about account status changes. The DS server needs an SMTP server to send messages, and needs templates for the mail it sends. By default, message templates are in English, and found in the `/path/to/opendj/config/messages/` directory.

DS servers generate notifications only when the server writes to an entry or evaluates a user entry for authentication. A server generates account enabled and account disabled notifications when the user account is enabled or disabled with the **manage-account** command, which writes to the entry. A server generates password expiration notifications when a user tries to bind.

For example, if you set up a server to send a notification about password expiration, that notification gets triggered when the user authenticates during the password expiration warning interval. The server does not automatically scan entries to send password expiry notifications. DS servers do implement controls that you can pass in an LDAP search to determine whether a user's password is about to expire. See "*LDAP Controls*" in the *Reference* for a list. You can send notifications based on the results of your search.

To Mail Users About Account Status

The following steps demonstrate how to set up notifications. Whether a server sends notifications depends on the settings in the password policy, and on account activity as described above.

1. Identify the SMTP server to receive the messages:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set smtp-server:opendj.example.com:25 \
 --trustAll \
 --no-prompt
```

2. Set up the DS server to be able to mail users about account status.

   The following example configures the server to send text-format mail messages:

```
$ dsconfig \
 set-account-status-notification-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name "SMTP Handler" \
 --set enabled:true \
 --set email-address-attribute-type:mail \
 --trustAll \
 --no-prompt
```

   Notice that the server finds the user's mail address on the attribute on the user's entry, specified by `email-address-attribute-type`.

   You can also configure the `message-subject` and `message-template-file` properties. Try interactive mode if you plan to do so.

   You find templates for messages by default under the `config/messages` directory. You can edit the templates to suit your purposes.

   If you edit the templates to send HTML rather than text messages, then set the advanced property, `send-email-as-html`, as shown in the following example:

```
$ dsconfig \
 set-account-status-notification-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name "SMTP Handler" \
 --set enabled:true \
 --set send-email-as-html:true \
 --trustAll \
 --no-prompt
```

3. Adjust applicable password policies to use the account status notification handler you configured:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set account-status-notification-handler:"SMTP Handler" \
 --trustAll \
 --no-prompt
```

About Notification Message Templates

When editing the `config/messages` templates, you can use the following tokens to have the server update the message text dynamically:

`%%notification-type%%`

This token is replaced with the name of the account status notification type for the notification.

`%%notification-message%%`

This token is replaced with the message for the account status notification.

`%%notification-user-dn%%`

This token is replaced with the string representation of the DN for the user who is the target of the account status notification.

`%%notification-user-attr:`*`attrname`*`%%`

This token is replaced with the value of the attribute specified by *attrname* from the user's entry. If the specified attribute has multiple values, then the DS server uses the first value encountered. If the specified attribute does not have any values, then the server replaces it with an emtpy string.

`%%notification-property:*propname*%%`

This token is replaced with the value of the specified notification property from the account status notification. If the specified property has multiple values, then the DS server uses the first value encountered. If the specified property does not have any values, then the server replaces it with an empty string. Valid *propname* values include the following:

- `account-unlock-time`
- `new-password`
- `old-password`
- `password-expiration-time`
- `password-policy-dn`
- `seconds-until-expiration`
- `seconds-until-unlock`
- `time-until-expiration`
- `time-until-unlock`

Chapter 12
Setting Resource Limits

This chapter shows you how to set resource limits that prevent directory clients from using an unfair share of system resources. In this chapter you will learn to:

- Limit the resources devoted to directory searches
- Limit concurrent client connections
- Restrict which clients can connect to the server
- Limit the amount of time connections can remain idle before they are dropped
- Limit the size of client requests
- Understand how resource limits are set for proxied authorization

Limiting Search Resources

Well-written directory client applications limit the search scope with filters that narrow the number of results returned. This prevents them from performing unindexed searches. By default, DS servers only allow users with appropriate privileges to perform unindexed searches.

In addition to letting the server prevent unindexed searches, you can set limits on search operations, such as the following:

- The *lookthrough limit* defines the maximum number of candidate entries that the DS server considers when processing a search.

  The default lookthrough limit of 5000 is set by the global server property `lookthrough-limit`.

  You can override the limit per user with the operational attribute, `ds-rlim-lookthrough-limit`.

- The *size limit* sets the maximum number of entries returned for a search.

  The default size limit of 1000 is set by the global server property `size-limit`.

  You can override the limit per user with the operational attribute, `ds-rlim-size-limit`.

  In addition, search requests themselves can include a size limit setting. The **ldapsearch** command has an `--sizeLimit` option.

- The *time limit* defines the maximum processing time the DS server devotes to a search operation.

  The default time limit of 1 minute is set by the global server property `time-limit`.

  You can override the limit on a per user basis with the operational attribute, `ds-rlim-time-limit`. Times for `ds-rlim-time-limit` are expressed in seconds.

  In addition, search requests themselves can include a time limit setting. The **ldapsearch** command has an `--timeLimit` option.

- The *idle time limit* defines how long the DS server allows idle connections to remain open.

  No default idle time limit is set. You can set an idle time limit by using the global server property `idle-time-limit`.

  You can override the limit on a per user basis with the operational attribute, `ds-rlim-idle-time-limit`. Times for `ds-rlim-idle-time-limit` are expressed in seconds.

- The maximum number of persistent searches is set by the global server property `max-psearches`.

This section includes the following procedures:

- "To Set Search Limits For a User"
- "To Set Search Limits For Users in a Group"
- "To Limit Concurrent Persistent Searches"

To Set Search Limits For a User

1. Give an administrator access to update the operational attributes related to search limits:

```
$ cat search-limits.ldif
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "ds-rlim-lookthrough-limit||ds-rlim-time-limit||ds-rlim-size-limit")
 (version 3.0;acl "Allow Kirsten Vaughan to manage search limits";
 allow (all) (userdn = "ldap:///uid=kvaughan,ou=People,dc=example,dc=com");)

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

2. Change the user entry to set the limits to override:

```
$ cat size-limit-bjensen.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: ds-rlim-size-limit
ds-rlim-size-limit: 10

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

When Babs Jensen performs a search returning more than 10 entries, she sees the following message:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 "(&)"
...
# The LDAP search request failed: 4 (Size Limit Exceeded)
# Additional Information:  This search operation has sent the maximum of 10 entries to the client
```

To Set Search Limits For Users in a Group

1. Give an administrator the privilege to write subentries, such as those used for setting collective attributes:

```
$ cat subentry-write.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: subentry-write

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

Notice here that the directory superuser, `cn=Directory Manager`, assigns privileges to Kirsten Vaughan. Any administrator with the `privilege-change` privilege can assign other privileges. Assuming the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, avoid assigning the `privilege-change` privilege to normal administrator users.

2. Create an LDAP subentry to specify the limits using collective attributes:

```
$ cat size-limit-collective.ldif
dn: cn=Remove Administrator Search Limits,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Remove Administrator Search Limits
ds-rlim-lookthrough-limit;collective: 0
ds-rlim-size-limit;collective: 0
ds-rlim-time-limit;collective: 0
subtreeSpecification: {base "ou=people", specificationFilter
  "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)" }

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

   The `base` entry identifies the branch that holds administrator entries. For details on how subentries apply, see "Understanding Subentry Scope".

3. Check the results:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN uid=kvaughan,ou=people,dc=example,dc=com \
 --searchScope base \
 "(&)" \
 ds-rlim-lookthrough-limit ds-rlim-time-limit ds-rlim-size-limit
ds-rlim-lookthrough-limit: 0
ds-rlim-time-limit: 0
ds-rlim-size-limit: 0
```

To Limit Concurrent Persistent Searches

An LDAP persistent search uses server resources in the following way. Each persistent search opens a connection and keeps it open, though the connection can be idle for long periods of time. When a modification changes data in the search scope, the server returns a search result to the persistent search client. The more concurrent persistent searches, the more work the server has to do for each modification:

- Set the global property `max-psearches` to limit the total number of concurrent persistent searches that the hDSserver accepts.

  The following example sets the limit to 30:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set max-psearches:30 \
 --trustAll \
 --no-prompt
```

Limiting and Restricting Client Connections

Each connection uses memory. On UNIX and Linux systems, each connection uses an available file descriptor. You can restrict which clients can connect to the server, and limit how many concurrent connections to the server are allowed.

Limit Total Concurrent Connections

To limit the total number of concurrent client connections that the server accepts, use the global setting `max-allowed-client-connections`. The following example sets the limit to 64K, which is the minimum number of file descriptors that should be available to the DS server:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set max-allowed-client-connections:65536 \
 --trustAll \
 --no-prompt
```

Restrict Which Clients Can Connect to the Server

To restrict which clients can connect to the server, use the global setting `allowed-client`, or `denied-client`. The following example restricts access to clients from the `example.com` domain:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set allowed-client:example.com \
 --trustAll \
 --no-prompt
```

You can set these properties to restrict which clients can connect on an individual `"Connection Handler"`. The settings on a connection handler override the global settings.

For details about how the server uses these settings, see "Limit Concurrent Connections Per Client".

Limit Concurrent Connections Per Client

To limit the number of concurrent connections from a client, use the global settings `restricted-client`, and `restricted-client-connection-limit`. The following example sets the limit for all clients on the `10.0.0.*` network to 1000 concurrent connections:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set restricted-client:"10.0.0.*" \
 --set restricted-client-connection-limit:1000 \
 --trustAll \
 --no-prompt
```

You can set these properties to limit per-client connections on an individual `"Connection Handler"`. The settings on a connection handler override the global settings.

The server uses the properties in this order:

1. If the `denied-client` property is set, the server denies connections from any client matching the settings.
2. If the `restricted-client` property is set, the server checks the number of connections from any client matching the settings.

   If a matching client exceeds `restricted-client-connection-limit` connections, the server refuses additional connections.
3. If the `allowed-client` property is set, the server allows connections from any client matching the settings.
4. If none of the properties are set, the server allows connections from any client.

Limiting Idle Time

If some client applications leave connections idle for long periods, DS servers can end up devoting resources to maintaining connections that are no longer used. If your network does not drop such connections eventually, you can configure the server to drop them by setting the global configuration property idle-time-limit. By default, no idle time limit is set.

If your network is configured to drop connections that have been idle for some time, set the DS idle time limit to a lower value than the idle time limit for the network. This helps to ensure that idle

connections are shut down in orderly fashion. Setting the DS limit lower than the network limit is particularly useful with networks that drop idle connections without cleanly closing the connection and notifying the client and server.

Note

DS servers do not enforce idle timeout for persistent searches.

The following example sets the idle-time-limit to 24 hours:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set idle-time-limit:24h \
 --trustAll \
 --no-prompt
```

Limiting Maximum Request Size

The default maximum request size of 5 MB is set using the advanced connection handler property `max-request-size`. This is sufficient for most deployments. In cases where clients add groups with large numbers of members, however, some add requests can exceed the 5 MB limit.

The following example increases the limit to 20 MB for the LDAP connection handler:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAP \
 --set max-request-size:20mb \
 --trustAll \
 --no-prompt
```

Note that this setting affects only the size of requests, not responses.

Resource Limits and Proxied Authorization

Proxied authorization uses a standard LDAP control to permit an application to bind as one user and then carry out LDAP operations on behalf of other users.

When using proxied authorization as described in "Configuring Proxied Authorization" in the *Developer's Guide*, be aware that the resource limits do not change when the user proxies as another user. In other words, resource limits depend on the bind DN, not the proxy authorization identity.

Chapter 13

Implementing Attribute Value Uniqueness

This chapter shows you how to enforce that specified attributes do not have repeated values in different directory entries. You can use attribute uniqueness, for example, to prevent two user entries sharing the same email address. In this chapter you will learn to:

- Enforce uniqueness for user IDs and other attributes
- Limit the scope of attribute value uniqueness
- Manage attribute value uniqueness across replicated directory servers

Some attribute values ought to remain unique. If you are using `uid` values as RDNs to distinguish between millions of user entries stored under `ou=People`, then you do not want your directory to contain two or more identical `uid` values. If your credit card or mobile number is stored as an attribute value on your directory entry, you certainly do not want to share that credit card or mobile number with another customer. The same is true for your email address.

The difficulty for you as directory administrator lies in implementing attribute value uniqueness without sacrificing the high availability that comes from using the servers's loosely consistent, multi-master data replication. Indeed, the DS replication model lets you maintain write access during network outages for directory applications. Yet, write access during a network outage can result in the same, theoretically unique attribute value getting assigned to two different entries at once. You do not notice the duplicate assignment until the network outage ends and replication resumes.

This chapter shows you how to set up attribute value uniqueness in your directory environment with the following procedures:

- "To Enable Unique UIDs"
- "To Enable Unique Values For Other Attributes"
- "To Limit The Scope of Uniqueness"
- "To Ensure Unique Attribute Values With Replication"

DS servers use the unique attribute plugin to handle attribute value uniqueness. As shown in the examples in this chapter, you can configure one unique attribute plugin to ensure uniqueness for multiple attributes and for entries under multiple base DNs. You can also configure multiple instances of the plugin for the same DS server.

To Enable Unique UIDs

DS servers provide a unique attribute plugin that you configure with the **dsconfig** command. By default, the plugin is prepared to ensure attribute values are unique for `uid` attributes.

1. Set the base DN where `uid` should have unique values, and enable the plugin:

```
$ dsconfig \
 set-plugin-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "UID Unique Attribute" \
 --set base-dn:ou=people,dc=example,dc=com \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

   Alternatively, you can specify multiple base DNs for unique values across multiple suffixes:

```
$ dsconfig \
 set-plugin-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "UID Unique Attribute" \
 --set enabled:true \
 --add base-dn:ou=people,dc=example,dc=com \
 --add base-dn:ou=people,dc=example,dc=org \
 --trustAll \
 --no-prompt
```

2. Check that the plugin is working correctly:

```
$ cat change-ajensen.ldif
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
add: uid
uid: bjensen

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 change-ajensen.ldif
# The LDAP modify request failed: 19 (Constraint Violation)
# Additional Information:  A unique attribute conflict was detected for attribute uid: value bjensen
 already exists in entry uid=bjensen,ou=People,dc=example,dc=com
```

If you have set up multiple suffixes, you can try something like this:

```
$ cat bjensen-org.ldif
dn: uid=bjensen,ou=People,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Babs
sn: Jensen
uid: bjensen

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 bjensen-org.ldif
# The LDAP modify request failed: 19 (Constraint Violation)
# Additional Information:  A unique attribute conflict was detected for attribute uid: value bjensen
 already exists in entry uid=bjensen,ou=People,dc=example,dc=com
```

To Enable Unique Values For Other Attributes

You can also configure the unique attribute plugin for use with other attributes, such as `mail`, `mobile`, or attributes you define, for example `cardNumber`.

1. Before you set up the plugin, index the attribute for equality.

   See "Configuring and Rebuilding Indexes" for instructions.

2. Set up the plugin configuration for your attribute.

   You can either add the attribute to an existing plugin configuration, or create a new plugin configuration including the attribute.

   When choosing between alternatives, be aware that values must be unique across the attributes and base DNs specified in each plugin configuration. Only group attributes in the same configuration if you want each value to be unique for all attributes. For example, you might create a single plugin configuration for telephone, fax, mobile, and pager numbers. As an alternative example, suppose user IDs are numeric, that user entries also specify `uidNumber`, and that user IDs are normally the same as their `uidNumber`s. In that case, create separate unique attribute configurations for `uid` and `uidNumber`.

   Apply one of these alternatives:

   - If you want to add the attribute to an existing plugin configuration, do so as shown in the following example which uses the plugin configuration from "To Enable Unique UIDs":

```
$ dsconfig \
 set-plugin-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "UID Unique Attribute" \
 --add type:telephoneNumber \
 --trustAll \
 --no-prompt
```

- If you want to create a new plugin configuration, do so as shown in the following example:

```
$ dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Unique phone numbers" \
 --type unique-attribute \
 --set enabled:true \
 --set base-dn:ou=people,dc=example,dc=com \
 --set type:telephoneNumber \
 --trustAll \
 --no-prompt
```

3. Check that the plugin is working correctly:

```
$ cat telephone-numbers.ldif
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
replace: telephoneNumber
telephoneNumber: +1 828 555 1212

dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: telephoneNumber
telephoneNumber: +1 828 555 1212

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 telephone-numbers.ldif
# MODIFY operation successful for DN uid=ajensen,ou=People,dc=example,dc=com

# The LDAP modify request failed: 19 (Constraint Violation)
# Additional Information:  A unique attribute conflict was detected for attribute telephoneNumber:
 value +1 828 555 1212 already exists in entry uid=ajensen,ou=People,dc=example,dc=com
```

To Limit The Scope of Uniqueness

In some cases you need attribute uniqueness separately for different base DNs in your directory. For example, you need all `uid` values to remain unique both for users in `dc=example,dc=com` and `dc=example,dc=org`, but it is not a problem to have one entry under each base DN with the same user ID as the organizations are separate. The following steps demonstrate how to limit the scope of uniqueness by creating separate configuration entries for the unique attribute plugin.

1. If the attribute you target is not indexed for equality by default, index the attribute for equality.

   See "Configuring and Rebuilding Indexes" for instructions.

   The examples in this procedure target the user ID attribute, `uid`, which is indexed for equality by default.

2. For each base DN, set up a configuration entry that ensures the target attribute values are unique:

```
$ dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Unique Example.com UIDs" \
 --type unique-attribute \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --set type:uid \
 --trustAll \
 --no-prompt
$ dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Unique Example.org UIDs" \
 --type unique-attribute \
 --set enabled:true \
 --set base-dn:dc=example,dc=org \
 --set type:uid \
 --trustAll \
 --no-prompt
```

3. Check that the plugin is working correctly:

```
$ cat unique-ids.ldif
dn: uid=unique,ou=People,dc=example,dc=com
uid: unique
givenName: Unique
```

```
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Unique Person
sn: Person
userPassword: 1Mun1qu3

dn: uid=unique,ou=People,dc=example,dc=org
uid: unique
givenName: Unique
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Unique Person
sn: Person
userPassword: 1Mun1qu3

dn: uid=copycat,ou=People,dc=example,dc=com
uid: unique
uid: copycat
givenName: Copycat
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Copycat Person
sn: Person
userPassword: copycopy

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 unique-ids.ldif
# Processing ADD request for uid=unique,ou=People,dc=example,dc=com
# ADD operation successful for DN uid=unique,ou=People,dc=example,dc=com
# Processing ADD request for uid=unique,ou=People,dc=example,dc=org
# ADD operation successful for DN uid=unique,ou=People,dc=example,dc=org
# Processing ADD request for uid=copycat,ou=People,dc=example,dc=com
# ADD operation failed
# The LDAP modify request failed:  19 (Constraint Violation)
# Additional Information:  A unique attribute conflict was detected for attribute uid: value unique
 already exists in entry uid=unique,ou=People,dc=example,dc=com
```

To Ensure Unique Attribute Values With Replication

The unique attribute plugin ensures unique attribute values on the DS directory server where the attribute value is updated. If client applications write the same attribute value separately at the same time on different replicas, it is possible that both servers consider the duplicate value unique, especially if the network is down between the replicas.

1. Enable the plugin identically on all replicas.

2. To avoid duplicate values where possible, use DS directory proxy to direct all updates of the unique attribute to the same directory server.

Chapter 14

Managing Schema

This chapter describes how to manage Lightweight Directory Access Protocol (LDAP) schema definitions for directory data. In this chapter you will learn to:

- Understand LDAP schemas including the schema definitions delivered with DS servers
- Change and extend DS LDAP schemas
- Relax schema checking when troubleshooting data that does not conform to schema definitions

Schema definitions describe the data, and especially the object classes and attribute types that can be stored in the directory. By default, DS servers conform strictly to LDAPv3 standards pertaining to schema definitions and attribute syntax checking, ensuring that data stored is valid and properly formed. Unless your data uses only standard schema present in the server when you install, then you must add additional schema definitions to account for the data your applications stored.

DS servers ship with many standard schema definitions. In addition, you can update and extend schema definitions while DS servers are online. As a result you can add new applications requiring additional data without stopping your directory service.

About Directory Schema

Directory schema, described in RFC 4512, defines the kinds of information you find in the directory, and can define how the information are related. This chapter focuses primarily on the following types of directory schema definitions:

- *Attribute type* definitions describe attributes of directory entries, such as `givenName` or `mail`.

  Here is an example of an attribute type definition:

  ```
  # Attribute type definition
  attributeTypes: ( 0.9.2342.19200300.100.1.3 NAME ( 'mail' 'rfc822Mailbox' )
    EQUALITY caseIgnoreIA5Match SUBSTR caseIgnoreIA5SubstringsMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{256} X-ORIGIN 'RFC 4524' )
  ```

  Attribute type definitions start with an OID, and generally a short name or names that are easier to remember than the OID. The attribute type definition can specify how attribute values should be collated for sorting, and what syntax they use. The X-ORIGIN is an extension to identify where the

definition originated. When you define your own schema, you likely want to provide an X-ORIGIN to help you to track versions of definitions, and where the definitions came from.

Attribute type definitions indicate whether the attribute is a *user attribute* intended to be modified by external applications (the default, or specified with `USAGE userApplications`), or a *operational attribute* intended to be managed by the server for internal purposes (specified, for example, with `USAGE directoryOperation`). The user attributes that are required and permitted on each entry are defined by the entry's object classes. The server checks what the entry's object classes require and permit when it handles requests to update user attributes. No such check is performed for operational attributes.

Attribute type definitions differentiate operational attributes with the following `USAGE` types:

- `USAGE directoryOperation` indicates a generic operational attribute.

  This is most likely the type of operational attribute that you would define if necessary. This is the type used, for example, when creating a last login time attribute in "Track Last Login Time".

- `USAGE dSAOperation` indicates a DSA-specific operational attribute, meaning an operational attribute specific to the current server.

- `USAGE distributedOperation` indicates a DSA-shared operational attribute, meaning an operational attribute shared by multiple servers.

- *Object class* definitions identify the attribute types that an entry must have, and may have. Examples of object classes include `person` and `organizationalUnit`.

  Here is an example of an object class definition:

  ```
  # Object class definition
  objectClasses: ( 2.5.6.6 NAME 'person' SUP top STRUCTURAL MUST ( sn $ cn )
    MAY ( userPassword $ telephoneNumber $ seeAlso $ description )
    X-ORIGIN 'RFC 4519' )
  ```

  Entries all have an attribute identifying their object classes, called `objectClass`.

  Object class definitions start with an object identifier (OID), and generally a short name that is easier to remember than the OID. The definition here says that the person object class inherits from the top object class, which is the top-level parent of all object classes. An entry's `objectclass` attribute lists the entry's object classes. An entry can have one STRUCTURAL object class inheritance branch, such as `top` - `person` - `organizationalPerson` - `inetOrgPerson`. Yet entries can have multiple AUXILIARY object classes. The object class then defines the attribute types that must be included, and the attribute types that may be included on entries having the object class.

- An *attribute syntax* constrains what directory clients can store as attribute values.

  An attribute syntax is identified in an attribute type definition by its OID. String-based syntax OIDs are optionally followed by a number set between braces that represents a minimum upper bound on the number of characters in the attribute value. For example, in the attribute type definition

shown above, the syntax is `1.3.6.1.4.1.1466.115.121.1.26{256}`. The syntax is an IA5 string (composed of characters from the international version of the ASCII character set) that can contain at least 256 characters.

You can find a table matching attribute syntax OIDs with their human-readable names in RFC 4517, Appendix A. Summary of Syntax Object Identifiers. The RFC describes attribute syntaxes in detail. You can list them by using the **dsconfig** command.

Although attribute syntaxes are often specified in attribute type definitions, DS servers do not always check that attribute values comply with attribute syntaxes. An DS server does tend to enforce compliance by default, in particular for certificates, country strings, directory strings, JPEG photos, and telephone numbers. The aim is to avoid accumulating garbage in your directory data.

If you are trying unsuccessfully to import non-compliant data from a more lenient directory server, you can either clean the data before importing it, or if cleaning the data is not an option, read "Relaxing Schema Checking to Import Legacy Data".

When creating your own attribute type definitions, use existing attribute syntaxes where possible. If you must create your own attribute syntax, then consider the extensions in Extensions for Attribute Syntax Descriptions.

- Matching rules determine how the DS server compares attribute values to assertion values for LDAP search and LDAP compare operations.

  For example, suppose you search with the filter `(uid=bjensen)`. The assertion value in this case is `bjensen`.

  DS servers have the following schema definition for the user ID attribute:

  ```
  attributeTypes: ( 0.9.2342.19200300.100.1.1 NAME ( 'uid' 'userid' )
   EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} X-ORIGIN 'RFC 4519' )
  ```

  When finding an equality match for your search, servers use the `caseIgnoreMatch` matching rule to check for user ID attribute values that equal `bjensen` without regard to case.

  You can read the schema definitions for matching rules that the server supports by performing an LDAP search, as shown in the following example:

  ```
  $ ldapsearch \
   --port 1389 \
   --baseDn cn=schema \
   --searchScope base \
   "(&)" \
   matchingRules
  ```

  Notice that many matching rules support string collation in languages other than English. For the full list of string collation matching rules, see "Directory Support For Locales and Language

Subtypes" in the *Reference*. For the list of other matching rules, see "*Matching Rules*" in the *LDAP Schema Reference*.

As you can read in examples such as "Search: Listing Active Accounts" in the *Developer's Guide*, matching rules enable directory clients to compare other values besides strings, for example.

DS servers expose schema over protocol through the `cn=schema` entry. The server stores the schema definitions corresponding to the entry in LDIF format files in the `db/schema/` directory. Many standard definitions and definitions pertaining to the server configuration are included at setup time.

Updating Directory Schema

DS servers are designed to permit updating the list of directory schema definitions while the server is running. As a result you can add support for new applications that require new attributes or new kinds of entries without interrupting the directory service. DS servers also replicate schema definitions, so the schema you add on one replica is propagated to other replicas without the need for manual intervention.

You can update schema using the **ldapmodify** command. Alternatively, create a schema file using a text editor, and add the file to the `db/schema/` directory before starting the server. The following example updates the schema to define a custom attribute type and a custom object class that uses the attribute:

```
$ cat 99-user-mods.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( temporary-fake-attr-id
  NAME 'myCustomAttribute'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications )
-
add: objectClasses
objectClasses: ( temporary-fake-oc-id
  NAME 'myCustomObjClass'
  SUP top
  AUXILIARY
  MAY myCustomAttribute )

$ cd /path/to/opendj/db/schema
$ cat 99-user.ldif
dn: cn=schema
objectClass: top
objectClass: ldapSubentry
objectClass: subschema
cn: schema
attributeTypes: ( temporary-fake-attr-id
  NAME 'myCustomAttribute'
  EQUALITY caseIgnoreMatch
```

```
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications )
objectClasses: ( temporary-fake-oc-id
  NAME 'myCustomObjClass'
  SUP top
  AUXILIARY
  MAY myCustomAttribute )
```

Make sure that you define schema elements before referencing them in other definitions. The server reads the schema files in sorted order, consuming schema definitions in the files as they occur. If, for example, a server reads a schema definition for an object class before the definitions of the attribute types used in the object class definition, then it displays an error. Your schema file should appear in the sorted list of file names *after* all the schema files that your schema definitions depends on.

Notice in the example that the schema changes show up by default in a file named `db/schema/99-user.ldif`. The file name starts with a number, 99. This number is larger than the numbers prefixing other schema file names. The default file name for custom, user-defined schema, `99-user.ldif`, ensures that those definitions load only after all of the schema files installed by default.

To test your schema definition, add the object class and attribute to an entry:

```
$ cat custom-attr.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: myCustomObjClass
-
add: myCustomAttribute
myCustomAttribute: Testing 1, 2, 3...

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 custom-attr.ldif
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 myCustomAttribute
dn: uid=bjensen,ou=People,dc=example,dc=com
myCustomAttribute: Testing 1, 2, 3...
```

You express schema definitions in LDIF. You can therefore update schema definitions as you would other directory data. For additional examples, see "Updating the Directory" in the *Developer's Guide*.

In addition to supporting the standard schema definitions that are described in RFC 4512, section 4.1, DS servers support the following extensions that you can use when adding your own definitions:

Extensions for All Schema Definitions

`X-DEPRECATED-SINCE`

Used to specify a release deprecating the schema element. Example: `X-DEPRECATED-SINCE: 6.5.6`

`X-ORIGIN`

Used to specify the origin of a schema element. Examples include `X-ORIGIN 'RFC 4519'`, `X-ORIGIN 'draft-ietf-ldup-subentry'`, and `X-ORIGIN 'DS Directory Server'`.

`X-SCHEMA-FILE`

Used to specify the relative path to the schema file containing the schema element such as `X-SCHEMA-FILE '00-core.ldif'`. Schema definitions are located by default in `/path/to/opendj/db/schema/*.ldif` files.

`X-STABILITY`

Used to specify the interface stability of the schema element.

This extension takes one of the following values:

`Evolving`
`Internal`
`Removed`
`Stable`
`Technology Preview`

For detailed descriptions, see "ForgeRock Product Stability Labels" in the *Release Notes*.

Extensions for Attribute Syntax Descriptions

Extensions to syntax definitions requires additional code to support syntax checking. DS servers support the following extensions for their particular use cases:

`X-ENUM`

Used to define a syntax that is an enumeration of values. The following attribute syntax description defines a syntax allowing four possible attribute values, for example:

```
ldapSyntaxes: ( security-label-syntax-oid DESC 'Security Label'
 X-ENUM ( 'top-secret' 'secret' 'confidential' 'unclassified' ) )
```

X-PATTERN

Used to define a syntax based on a regular expression pattern, where valid regular expressions are those defined for `java.util.regex.Pattern`. The following attribute syntax description defines a simple, lenient SIP phone URI syntax check:

```
ldapSyntaxes: ( simple-sip-uri-syntax-oid DESC 'Lenient SIP URI Syntax'
 X-PATTERN '^sip:[a-zA-Z0-9.]+@[a-zA-Z0-9.]+(:[0-9]+)?$' )
```

X-SUBST

Used as a fallback to substitute a defined syntax for one that DS servers do not implement. The following example substitutes Directory String syntax, which has OID 1.3.6.1.4.1.1466.115.121.1.15, for a syntax that DS servers do not implement:

```
ldapSyntaxes: ( non-implemented-syntax-oid DESC 'Not Implemented in DS'
 X-SUBST '1.3.6.1.4.1.1466.115.121.1.15' )
```

Extension for Attribute Type Descriptions

X-APPROX

`X-APPROX` is used to specify the approximate matching rule to use for a given attribute type when not using the default, which is the double metaphone approximate match.

Working With JSON

DS software has the following features for working with JSON objects:

- **RESTful HTTP access to directory services**

  If you have LDAP data, but HTTP client applications want JSON over HTTP instead, DS software can expose your LDAP data as JSON resources over HTTP to REST clients. As described in "About RESTful Access to Directory Services", you can configure how LDAP entries map to JSON resources.

  There is no requirement to change LDAP schema definitions before using this feature. To get started, see "RESTful Client Access Over HTTP", and "*Performing RESTful Operations*" in the *Developer's Guide*.

- **JSON syntax LDAP attributes**

  If you have LDAP client applications that store JSON in the directory, you can define LDAP attributes that have `Json` syntax.

The following schema excerpt defines an attribute called `json` with case-insensitive matching:

```
attributeTypes: ( json-attribute-oid NAME 'json'
 SYNTAX 1.3.6.1.4.1.36733.2.1.3.1 EQUALITY caseIgnoreJsonQueryMatch
 SINGLE-VALUE X-ORIGIN 'DS Documentation Examples' )
```

Notice that the JSON syntax OID is `1.3.6.1.4.1.36733.2.1.3.1`. The definition above uses the (default) `caseIgnoreJsonQueryMatch` matching rule for equality. As explained later in this section, you might want to choose different matching rules for your JSON attributes.

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

- **Configurable indexing for JSON attributes**

  When you store JSON attributes in the directory, you can index every field in each JSON attribute value, or you can index only what you use.

  As for other LDAP attribues, the indexes depend on the matching rule defined for the JSON syntax attribute.

DS servers treat JSON syntax attribute values as objects. Two JSON values may be considered equivalent despite differences in their string representations. The following JSON objects can be considered equivalent because each field has the same value. Their string representations are different, however:

```
{ "id": "bjensen", "given-name": "Barbara", "surname": "Jensen" }
{"surname":"Jensen","given-name":"Barbara","id":"bjensen"}
```

Unlike other objects with their own LDAP attribute syntaxes, such as X.509 certificates, two JSON objects with completely different structures (different field names and types) are still both JSON. Nothing in the JSON syntax alone tells the server anything about what a JSON object must and may contain.

When defining LDAP schema for JSON attributes, it helps therefore to understand the structure of the expected JSON. Will the attribute values be arbitrary JSON objects, or JSON objects whose structure is governed by some common schema? If a JSON attribute value is an arbitrary object, you can do little to optimize how it is indexed or compared. If the value is a structured object, however, you can configure optimizations based on the structure.

For structured JSON objects, the definition of JSON object equality is what enables you to pick the optimal matching rule for the LDAP schema definition. The matching rule determines how the server indexes the attribute, and how the server compares two JSON values for equality. You can define equality in the following ways:

- Two JSON objects are equal if *all* fields have the same values.

By this definition, `{"a": 1, "b": true}` equals `{"b":true,"a":1}`. However, `{"a": 1, "b": true}` and `{"a": 1, "b": "true"}` are different.

- Two JSON objects are equal if *some* fields have the same values. Other fields are ignored when comparing for equality.

  For example, take the case where two JSON objects are considered equal if they have the same "_id" values. By this definition, `{"_id":1,"b":true}` equals `{"_id":1,"b":false}`. However, `{"_id":1,"b":true}` and `{"_id":2,"b":true}` are different.

  DS servers have built-in matching rules for the case where equality means "_id" values are equal. If the fields to compare are different from "_id", you must define your own matching rule and configure a custom schema provider that implements it. This is reflected in "Choosing JSON Equality Matching Rules".

Choosing JSON Equality Matching Rules

JSON Content	Two JSON Objects are Equal if...	Use One of These Matching Rules
Arbitrary (any valid JSON is allowed)	All fields have the same values.	`caseExactJsonQueryMatch` `caseIgnoreJsonQueryMatch`
Structured	All fields have the same values.	`caseExactJsonQueryMatch` `caseIgnoreJsonQueryMatch`
Structured	"_id" fields have the same values. Additional fields are ignored when comparing for equality.	`caseExactJsonIdMatch` `caseIgnoreJsonIdMatch`
Structured	One or more other fields have the same values. Additional fields are ignored when comparing for equality.	When using this matching rule, create a custom `json-equality-matching-rule` Schema Provider. The custom schema provider must include all the necessary properties and reference the custom field(s). See "Example: Index Using a Custom JSON Equality Matching Rule".

When you choose an equality matching rule in the LDAP attribute definition, you are also choosing the default that applies in an LDAP search filter equality assertion. For example, `caseIgnoreJsonQueryMatch` works with filters such as `"(json=id eq 'bjensen')"`. `caseIgnoreJsonIdMatch` works with filters such as `'(json={"_id":"bjensen"})'`.

DS servers also implement JSON ordering matching rules for determining the relative order of two JSON values using a custom set of rules. You can select which JSON fields should be used for performing the ordering match. You can also define whether those fields that contain strings should be normalized before comparison by trimming white space or ignoring case differences. DS servers can implement JSON ordering matching rules on demand when presented with an extended server-

side sort request as described in "Search: Server-Side Sort" in the *Developer's Guide*. If, however, you define them statically in your LDAP schema, then you must implement them by creating a custom `json-ordering-matching-rule` Schema Provider. For details about the `json-ordering-matching-rule` object's properties, see JSON Ordering Matching Rule.

For examples showing how to add LDAP schema for new attributes, see "Updating Directory Schema". For examples showing how to index JSON attributes, see "Configuring Indexes for JSON Attributes".

Relaxing Schema Checking to Import Legacy Data

By default, DS servers accept data that follows the schema for allowable and rejected data. You might have legacy data from a directory service that is more lenient, allowing non-standard constructions such as multiple structural object classes per entry, not checking attribute value syntax, or even not respecting schema definitions.

For example, when importing data with multiple structural object classes defined per entry, you can relax schema checking to warn rather than reject entries having this issue:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set single-structural-objectclass-behavior:warn \
 --trustAll \
 --no-prompt
```

You can allow attribute values that do not respect the defined syntax with the **dsconfig** command as well:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set invalid-attribute-syntax-behavior:warn \
 --trustAll \
 --no-prompt
```

You can even turn off schema checking altogether, although turning off schema checking only really makes sense when you are absolutely sure that the entries and attribute values respect the schema definitions, and you simply want to turn off schema checking temporarily to speed up import processing:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set check-schema:false \
 --trustAll \
 --no-prompt
```

Standard Schema Included With DS Servers

DS servers provide many standard schema definitions. For documentation on the available schema definitions, see LDAP Schema Reference. The LDAP schema elements are defined in these LDIF files in the `/path/to/opendj/db/schema/` directory:

`00-core.ldif`

This file contains a core set of attribute type and object class definitions from the following Internet-Drafts, RFCs, and standards:

draft-boreham-numsubordinates
draft-findlay-ldap-groupofentries
draft-good-ldap-changelog
draft-howard-namedobject
draft-ietf-ldup-subentry
draft-wahl-ldap-adminaddr
RFC 1274
RFC 2079
RFC 2256
RFC 2798
RFC 3045
RFC 3296
RFC 3671
RFC 3672
RFC 4512
RFC 4519
RFC 4523
RFC 4524
RFC 4530
RFC 5020
X.501

`01-pwpolicy.ldif`

This file contains schema definitions from draft-behera-ldap-password-policy (Draft 09), which defines a mechanism for storing password policy information in an LDAP directory server.

`02-config.ldif`

This file contains the attribute type and objectclass definitions for use with the server configuration.

`03-changelog.ldif`

This file contains schema definitions from draft-good-ldap-changelog, which defines a mechanism for storing information about changes to directory server data.

`03-rfc2713.ldif`

This file contains schema definitions from RFC 2713, which defines a mechanism for storing serialized Java objects in the directory server.

`03-rfc2714.ldif`

This file contains schema definitions from RFC 2714, which defines a mechanism for storing CORBA objects in the directory server.

`03-rfc2739.ldif`

This file contains schema definitions from RFC 2739, which defines a mechanism for storing calendar and vCard objects in the directory server. Note that the definition in RFC 2739 contains a number of errors, and this schema file has been altered from the standard definition in order to fix a number of those problems.

`03-rfc2926.ldif`

This file contains schema definitions from RFC 2926, which defines a mechanism for mapping between Service Location Protocol (SLP) advertisements and LDAP.

`03-rfc3112.ldif`

This file contains schema definitions from RFC 3112, which defines the authentication password schema.

`03-rfc3712.ldif`

This file contains schema definitions from RFC 3712, which defines a mechanism for storing printer information in the directory server.

`03-uddiv3.ldif`

This file contains schema definitions from RFC 4403, which defines a mechanism for storing UDDIv3 information in the directory server.

`04-rfc2307bis.ldif`

This file contains schema definitions from draft-howard-rfc2307bis, which defines a mechanism for storing naming service information in the directory server.

`05-rfc4876.ldif`

This file contains schema definitions from RFC 4876, which defines a schema for storing Directory User Agent (DUA) profiles and preferences in the directory server.

`05-samba.ldif`

This file contains schema definitions required when storing Samba user accounts in the directory server.

`05-solaris.ldif`

This file contains schema definitions required for Solaris and OpenSolaris LDAP naming services.

`06-compat.ldif`

This file contains the attribute type and objectclass definitions for use with the server configuration.

Chapter 15

Configuring REST APIs

As described in "To Set Up REST Access to User Data", you can expose LDAP data as JSON objects over HTTP. You define the mapping between the data representations, and expose the REST view through a Rest2ldap endpoint. In this chapter, you will learn how to:

- Add a REST to LDAP mapping for a custom object with custom attributes
- Add a mapping for subentry password policies
- Map JSON profile data to LDAP entries
- Generate OpenAPI reference documentation for your REST APIs

Before trying the examples in this chapter, set up a directory server with HTTP access as shown in the following command:

```
$ /path/to/opendj/setup \
 directory-server \
 --rootUserDn "cn=Directory Manager" \
 --rootUserPassword password \
 --hostname opendj.example.com \
 --ldapPort 1389 \
 --httpPort 8080 \
 --httpsPort 8443 \
 --adminConnectorPort 4444 \
 --profile ds-evaluation \
 --acceptLicense \
 --quiet
```

For examples demonstrating how to use the REST APIs, also see "*Performing RESTful Operations*" in the *Developer's Guide*.

Adding a REST to LDAP Mapping for a Custom Object

The example REST to LDAP mapping configuration file, `config/rest2ldap/endpoints/api/example-v1.json`, works well with sample data for evaluation. For most deployments, you must customize the mapping to expose additional information.

This section demonstrates how to configure an additional mapping for a custom LDAP object called `affiliateObject`, whose `affiliate` attribute references user entries. For full reference information concerning REST to LDAP mappings, see "*REST to LDAP Configuration*" in the *Reference*.

This demonstration uses a directory server set up for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*. The mappings will also work if you are using the REST to LDAP gateway.

To begin the demonstration, add the schema definitions for the custom LDAP object, and add an example LDAP entry using the object class. The following example adds the schema definitions, and then adds an example `My Affiliates` entry to the directory server:

```
$ cat 99-example-mods.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( affiliate-oid  NAME 'affiliate' SUP distinguishedName X-SCHEMA-FILE '99-example-
mods.ldif' )
-
add: objectClasses
objectClasses: ( affiliateObject-oid NAME 'affiliateObject' SUP top STRUCTURAL MUST cn MAY ( description $
 affiliate ) X-SCHEMA-FILE '99-example-mods.ldif' )

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 99-example-mods.ldif

$ cat add-affiliates.ldif
dn: ou=affiliates,dc=example,dc=com
objectClass: top
objectClass: organizationalUnit
ou: affiliates

dn: cn=My Affiliates,ou=affiliates,dc=example,dc=com
objectClass: top
objectClass: affiliateObject
cn: My Affiliates
affiliate: uid=bjensen,ou=People,dc=example,dc=com
affiliate: uid=kvaughan,ou=People,dc=example,dc=com

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

For details about custom LDAP schema definitions, see "*Managing Schema*". For details about adding entries to an LDAP directory service, see "Adding Entries" in the *Developer's Guide*.

In the REST to LDAP mapping configuration file, define an `affiliates` collection subresource, and an `examples:affiliates:1.0` resource type that describes the objects in the `affiliates` collection. This causes the REST API to add an `/api/affiliates` path next to `/api/users` and `/api/groups`. Under `/api/affiliates`, REST clients can access the affiliate entries as JSON resources.

The definition for the `affiliates` collection subresource belongs in the set of `subResources` in the mapping configuration file. Its definition is:

```
// This sub-resource definition maps JSON resources under /api/affiliates
// to LDAP entries under ou=affiliates,dc=example,dc=com.
"affiliates" : {
    "type": "collection",
    "dnTemplate": "ou=affiliates,dc=example,dc=com",
    "resource": "examples:affiliates:1.0",
    "namingStrategy": {
        "type": "clientDnNaming",
        "dnAttribute": "cn"
    }
}
```

Notice that the parser for the REST to LDAP mapping configuration file is lenient. It lets you include comments in the JSON, although the JSON standard does not allow comments.

The definition for the `examples:affiliates:1.0` resource type belongs in the set of `example-v1` resource types in the mapping configuration file. Its definition is:

```
// An "affiliate" resource includes property mappings for an "affiliateObject",
// which is identified by the "cn" attribute. The affiliate DNs reference person entries.
// Rather than return DNs in JSON resources, a mapper returns identifiers and display names.
"examples:affiliates:1.0": {
    "superType": "frapi:opendj:rest2ldap:object:1.0",
    "objectClasses": [ "affiliateObject" ],
    "properties": {
        "displayName": {
            "type": "simple",
            "ldapAttribute": "cn",
            "isRequired": true,
            "writability": "createOnly"
        },
        "affiliate": {
            "type": "reference",
            "baseDn": "ou=people,..,..",
            "isMultiValued": true,
            "primaryKey": "uid",
            "mapper": {
                "type": "object",
                "properties": {
                    "_id": {
                        "type": "simple",
                        "ldapAttribute": "uid",
                        "isRequired": true
                    },
                    "displayName": {
                        "type": "simple",
                        "ldapAttribute": "cn",
                        "writability": "readOnlyDiscardWrites"
                    }
                }
            }
        },
        "description": {
            "type": "simple"
        }
    }
}
```

To see these in context, download and review the augmented mapping configuration file, example-v1.json.

Copy the augmented mapping configuration file into the directory configuration file, `config/rest2ldap/endpoints/api/example-v1.json`. If you have not already done so, enable HTTP access as described in "To Set Up REST Access to User Data". The following example forces the Rest2ldap endpoint to reread its configuration by disabling it and then enabling it:

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

With the updated REST to LDAP mapping configuration taken into account, REST clients can access the affiliates resources, as shown in the following example:

```
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/affiliates/my%20affiliates?_fields=affiliate&_prettyPrint=true"
{
  "_id" : "My Affiliates",
  "_rev" : "<revision>",
  "affiliate" : [ {
    "_id" : "bjensen",
    "displayName" : [ "Barbara Jensen", "Babs Jensen" ]
  }, {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  } ]
}
```

Adding Subentry Password Policies

This section demonstrates how to configure REST to LDAP to manage subentry-based password policies. For background, see "*Configuring Password Policy*". For full reference information concerning REST to LDAP mappings, see "*REST to LDAP Configuration*" in the *Reference*.

This demonstration uses a directory server set up for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*. The mappings will also work if you are using the REST to LDAP gateway.

To begin the demonstration, edit the default REST to LDAP mapping file, `config/rest2ldap/endpoints/api/example-v1.json`.

Define an `/api/subentries` endpoint that exposes LDAP subentries over REST, including password policies, by adding the following under `subResources`:

```
// This subresource definition maps JSON resources under /api/subentries
// to subentries under dc=example,dc=com.
"subentries" : {
  "type": "collection",
  "dnTemplate": "dc=example,dc=com",
  "resource": "examples:subentry:1.0",
  "namingStrategy": {
    "type": "clientDnNaming",
    "dnAttribute": "cn"
  }
}
```

Define the subentry and password policy under the other `example-v1` resource types:

```
// A subentry resource maps to a subentry object.
"examples:subentry:1.0": {
  "superType": "frapi:opendj:rest2ldap:object:1.0",
  "isAbstract": true,
  "objectClasses": [ "top", "subentry" ],
  "properties": {
    "_id": {
      "type": "simple",
      "ldapAttribute": "cn",
      "isRequired": true,
      "writability": "createOnly"
    },
    "subtreeSpecification": {
      "type": "simple"
    }
  }
},
// A passwordPolicy resource maps to a subentry password policy object.
// This mapping uses the LDAP attribute names,
// except for passwordValidator which maps to the validator in the configuration.
"examples:passwordPolicy:1.0": {
  "superType": "examples:subentry:1.0",
  "objectClasses": [ "pwdPolicy", "pwdValidatorPolicy", "subentry" ],
  "properties": {
    "pwdAttribute": {
      "type": "simple",
```

```
      "isRequired": true
    },
    "pwdAllowUserChange": {
      "type": "simple"
    },
    "pwdCheckQuality": {
      "type": "simple"
    },
    "pwdExpireWarning": {
      "type": "simple"
    },
    "pwdFailureCountInterval": {
      "type": "simple"
    },
    "pwdGraceAuthNLimit": {
      "type": "simple"
    },
    "pwdInHistory": {
      "type": "simple"
    },
    "pwdLockout": {
      "type": "simple"
    },
    "pwdLockoutDuration": {
      "type": "simple"
    },
    "pwdMaxAge": {
      "type": "simple"
    },
    "pwdMaxFailure": {
      "type": "simple"
    },
    "pwdMinAge": {
      "type": "simple"
    },
    "pwdMinLength": {
      "type": "simple"
    },
    "pwdMustChange": {
      "type": "simple"
    },
    "pwdSafeModify": {
      "type": "simple"
    },
    "passwordValidator" : {
      "type": "reference",
      "baseDn": "cn=Password Validators,cn=config",
      "ldapAttribute": "ds-cfg-password-validator",
      "primaryKey": "cn",
      "isMultiValued": true,
      "mapper": {
        "type": "object",
        "properties": {
          "_id": {
            "type": "simple",
            "ldapAttribute": "cn",
            "isRequired": true
          }
        }
```

```
      }
    }
  }
```

In LDAP, you can edit user entries to assign password policies. For REST to LDAP, add corresponding password policy properties to the `frapi:opendj:rest2ldap:user:1.0` resource type:

```
// Administrators can read the current password policy.
"pwdPolicy": {
  "type": "reference",
  "baseDn": "..,..",
  "ldapAttribute": "pwdPolicySubentry",
  "primaryKey": "cn",
  "searchFilter": "(&(objectclass=subentry)(objectclass=pwdPolicy))",
  "mapper": {
    "type": "object",
    "properties": {
      "_id": {
        "type": "simple",
        "ldapAttribute": "cn",
        "writability": "readOnlyDiscardWrites"
      }
    }
  }
},
// Administrators can set a new password policy.
"newPwdPolicy": {
  "type": "reference",
  "baseDn": "..,..",
  "ldapAttribute": "ds-pwp-password-policy-dn",
  "primaryKey": "cn",
  "searchFilter": "(&(objectclass=subentry)(objectclass=pwdPolicy))",
  "mapper": {
    "type": "object",
    "properties": {
      "_id": {
        "type": "simple",
        "ldapAttribute": "cn",
        "isRequired": true
      }
    }
  }
},
```

To see these in context, download and review the augmented mapping configuration file, subentries.json.

Copy the augmented mapping configuration file into the directory configuration file, `config/rest2ldap/endpoints/api/example-v1.json`. If you have not already done so, enable HTTP access as described in "To Set Up REST Access to User Data". The following example forces the Rest2ldap endpoint to reread its configuration by disabling it and then enabling it:

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

With the augmented mapping in place, grant access for password administrators to edit and assign password policies:

- Grant the necessary privileges to edit subentry-based password policies:
  - The `subentry-write` privilege lets password administrators edit subentries.
  - The `config-read` privilege lets password administrators reference password validator entries stored in the server configuration.
- Grant access to assign password policies, letting password administrators read users' `pwdPolicySubentry` attributes and write users' `ds-pwp-password-policy-dn` attributes.
- Grant access to assign password policies, letting password administrators write the subentry attributes, `ds-pwp-password-validator` and `subtreeSpecification`.

For this demonstration, grant access to the administrator user Kirsten Vaughan. Kirsten's entry, `uid: kvaughan`, is created when you set up the directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*:

```
# Assign privileges:
$ cat config-read.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: config-read

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
```

```
 --bindPassword password \
 config-read.ldif

$ cat subentry-write.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: subentry-write

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 subentry-write.ldif

# Grant access:
$ cat edit-pwp.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "pwdPolicySubentry||ds-pwp-password-policy-dn||ds-pwp-password-validator||
subtreeSpecification")
 (version 3.0;acl "Allow Administrators to manage users' password policies";
 allow (all) (groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com");)

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

Create and assign a subentry-based password policy:

```
$ curl \
 --request PUT \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --header "If-None-Match: *" \
 --data '{
  "_id": "Subentry Password Policy with Validators",
  "_schema": "examples:passwordPolicy:1.0",
  "pwdAttribute": "userPassword",
  "pwdAllowUserChange": true,
  "pwdFailureCountInterval": 300,
  "pwdLockout": true,
  "pwdLockoutDuration": 300,
  "pwdMaxFailure": 3,
  "pwdSafeModify": true,
  "passwordValidator": [{"_id": "Character Set"}, {"_id": "Length-Based Password Validator"}],
  "subtreeSpecification": "{base \"ou=people\", specificationFilter \"(isMemberOf=cn=Directory
 Administrators,ou=Groups,dc=example,dc=com)\" }"
 }' \
 http://opendj.example.com:8080/api/subentries/Subentry%20Password%20Policy%20with%20Validators?
_prettyPrint=true
{
  "_id" : "Subentry Password Policy with Validators",
  "_rev" : "<revision>",
  "_schema" : "examples:passwordPolicies:1.0",
```

```
  "_meta" : {
    "created" : "<date>"
  },
  "pwdAttribute" : "userPassword",
  "pwdAllowUserChange" : true,
  "pwdFailureCountInterval" : 300,
  "pwdLockout" : true,
  "pwdLockoutDuration" : 300,
  "pwdMaxFailure" : 3,
  "pwdSafeModify" : true,
  "passwordValidator" : [ {
    "_id" : "Character Set"
  }, {
    "_id" : "Length-Based Password Validator"
  } ],
  "subtreeSpecification" : "{base \"ou=people\", specificationFilter \"(isMemberOf=cn=Directory
 Administrators,ou=Groups,dc=example,dc=com)\" }"
}
```

Observe that the password administrator can view which password policy applies:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users/kvaughan?_fields=pwdPolicy&_prettyPrint=true"
{
  "_id" : "kvaughan",
  "_rev" : "<revision>",
  "pwdPolicy" : {
    "_id" : "Subentry Password Policy with Validators"
  }
}
```

Observe that the field is not visible to regular users:

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

Mapping JSON Profiles to LDAP

This section demonstrates what to do when you start building a REST API based on JSON examples or JSON schemas, rather than LDAP schemas and LDAP entries.

This section develops a REST API for simple user profiles. The following JSON is a example profile:

```
{
    "externalId": "newuser",
    "userName": "newuser",
    "name": {
        "formatted": "New User",
        "givenName": "User",
        "familyName": "New"
    },
    "phoneNumbers": [{
        "value": "+1 408 555 1212",
        "type": "work"
    }],
    "emails": [{
            "value": "newuser@example.com",
            "type": "work",
            "primary": true
        },
        {
            "value": "newuser@example.org",
            "type": "personal",
            "primary": false
        }
    ]
}
```

Notice these key features of the example profile:

- Unlike Common REST JSON objects, this profile has no `"_id"` field.
- The `"phoneNumbers"` and `"emails"` fields are multi-valued, similar to the LDAP `telephoneNumber` and `mail` attributes.

  However, both JSON fields hold arrays of nested objects. The similar LDAP attributes hold simple values. The other fields have a straightforward mapping to standard LDAP attributes, but these fields do not.

  The `"type"` field in a phone number or email is an arbitrary label assigned by the user.

Representing JSON in LDAP

This example assumes that the user profile does not exist in LDAP yet. The aim is therefore to translate JSON objects into LDAP entries.

LDAP attributes generally hold values with tightly defined syntaxes, not arbitrary objects. JSON syntax attributes, described in "Working With JSON", are an exception. The example profile holds a mix of fields that map directly to LDAP attributes, and fields that do not.

When determining them LDAP attributes to use, the first step is to look for natural matches with attributes defined in the default schema. Based on the *LDAP Schema Reference*, you can derive the following table of correspondences:

Profile JSON Field	LDAP Attribute
`"externalId"`	`uid`
`"userName"`	`uid`
`"name/formatted"`	`cn`
`"name/givenName"`	`givenName`
`"name/familyName"`	`sn`
`"emails"`	No direct match.
`"phoneNumbers"`	No direct match.

For the `"emails"` and `"phoneNumbers"` fields, there is no natural match. The `uddiEMail` and `uddiPhone` attributes come the closest because they optionally include a user-defined type. However, avoid those attributes for the following reasons:

- These user profiles are unrelated to Universal Description, Discovery, and Integration (UDDI), which is focused instead on storing data for a SOAP-based web services discovery.
- All client applications would have to know how to consume the UDDI-specific format. REST to LDAP has no means to transform *`type#value`* into `{"type": "type", "value": "value"}`.
- The `uddiEmail` has no provision for the `"primary"` boolean value.

It would also be possible, but not advisable, to store `"emails"` and `"phoneNumbers"` in separate LDAP entries. The LDAP user entry could include attributes that reference email address and phone number entries by their DNs. The email and phone number entries could in turn reference users with the standard `owner` attribute. Unfortunately, there is no real value in storing email addresses and phone numbers separately from a user's entry. The email and phone settings will not be shared with other entries, but instead belong only to the profile. Client applications will expect to update them atomically with the user profile. A profile change should not require updating an arbitrary number of LDAP entries. Even if they could be conveniently configured as single updates in the REST API, it would mean that updates to a JSON resource could partially fail in LDAP, a source of potential confusion.

An apt choice for the `"emails"` and `"phoneNumbers"` fields is therefore JSON syntax attributes. This example defines the following LDAP schema definitions for appropriate JSON attributes:

```
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-email-oid
  NAME 'email'
  DESC 'An email address with an optional user-defined type and primary boolean'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-phone-number-oid
  NAME 'phoneNumber'
  DESC 'A phone number with an optional user-defined type'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-profile-user-oid
  NAME 'profileUser'
  DESC 'A user with optional profile components having user-defined types'
  SUP top
  AUXILIARY
  MAY ( email $ phoneNumber )
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
```

An auxiliary object class lets an entry that already has a structural object class hold additional attributes. For background information about LDAP schema, read "*Managing Schema*".

The following corresponding LDIF uses these definitions to define Example.com data with one user profile. Babs Jensen is the administrator and sole initial user. She can create other profiles and reset passwords:

```
dn: dc=example,dc=com
objectClass: domain
objectClass: top
dc: example
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr != "userPassword")
 (version 3.0;acl "Authenticated users have read-search access";
 allow (read, search, compare)(userdn = "ldap:///all");)
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr = "*")
 (version 3.0;acl "Allow Babs to administer other entries";
 allow (all)(userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)

dn: ou=People,dc=example,dc=com
objectClass: organizationalunit
objectClass: top
ou: People
aci: (target ="ldap:///ou=People,dc=example,dc=com")
```

```
 (targetattr = "*")
 (version 3.0; acl "Allow self management of profiles";
 allow (delete, write)(userdn = "ldap:///self");)

dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: profileUser
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
uid: bjensen
ou: People
cn: Barbara Jensen
cn: Babs Jensen
givenname: Barbara
sn: Jensen
userpassword: hifalutin
email: {"value": "bjensen@example.com", "type": "work", "primary": true}
phoneNumber: {"value": "+1 408 555 1862", "type": "work"}
ds-privilege-name: password-reset
```

For additional background information on the `aci` and `ds-privilege-name` attributes used here, read "*Configuring Privileges and Access Control*".

Indexing the JSON Attributes

Client applications might look up user profiles based on email addresses or phone numbers. They are not likely to look up user profiles based on user-defined labels for emails and phone numbers. Therefore, instead of indexing all fields of each JSON attribute value, you can index only the values. Preparing the index is described in "Configuring Indexes for JSON Attributes".

To Add Indexes for the JSON Attributes

The following steps demonstrate how to prepare the appropriate indexes for this example:

1. If you have not already done so, set up a directory server with HTTP and HTTPS access:

   ```
   $ /path/to/opendj/setup \
    directory-server \
    --rootUserDn "cn=Directory Manager" \
    --rootUserPassword password \
    --hostname opendj.example.com \
    --ldapPort 1389 \
    --httpPort 8080 \
    --httpsPort 8443 \
    --adminConnectorPort 4444 \
    --profile ds-evaluation \
    --acceptLicense \
    --quiet
   ```

   Alternatively, you can use the REST to LDAP gateway for HTTP access. This example uses embedded HTTP connection handlers to simplify installation and configuration.

2. Configure a custom schema provider to allow the server to index only the JSON `"values"`:

```
$ dsconfig \
 create-schema-provider \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Value JSON Query Matching Rule" \
 --type json-query-equality-matching-rule \
 --set enabled:true \
 --set case-sensitive-strings:false \
 --set ignore-white-space:true \
 --set matching-rule-name:caseIgnoreJsonQueryMatch \
 --set matching-rule-oid:1.3.6.1.4.1.36733.2.1.4.1 \
 --set indexed-field:value \
 --trustAll \
 --no-prompt
```

3. Update the LDAP schema defined in "Representing JSON in LDAP":

```
$ cat 99-example.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-email-oid
  NAME 'email'
  DESC 'An email address with an optional user-defined type and primary boolean'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-phone-number-oid
  NAME 'phoneNumber'
  DESC 'A phone number with an optional user-defined type'
  EQUALITY caseIgnoreJsonQueryMatch
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.1
  USAGE userApplications
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-profile-user-oid
  NAME 'profileUser'
  DESC 'A user with optional profile components having user-defined types'
  SUP top
  AUXILIARY
  MAY ( email $ phoneNumber )
  X-SCHEMA-FILE '99-example.ldif'
  X-ORIGIN 'DS Documentation Examples' )
$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 99-example.ldif
```

4. Add indexes for the `email` and `phoneNumber` JSON attributes:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name email \
 --set index-type:equality \
 --trustAll \
 --no-prompt
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name phoneNumber \
 --set index-type:equality \
 --trustAll \
 --no-prompt
```

5. Import the LDAP data with Babs's profile shown in "Representing JSON in LDAP":

```
$ cat profiles.ldif
dn: dc=example,dc=com
objectClass: domain
objectClass: top
dc: example
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr != "userPassword")
 (version 3.0;acl "Authenticated users have read-search access";
 allow (read, search, compare)(userdn = "ldap:///all");)
aci: (target ="ldap:///dc=example,dc=com")
 (targetattr = "*")
 (version 3.0;acl "Allow Babs to administer other entries";
 allow (all)(userdn = "ldap:///uid=bjensen,ou=People,dc=example,dc=com");)

dn: ou=People,dc=example,dc=com
objectClass: organizationalunit
objectClass: top
ou: People
aci: (target ="ldap:///ou=People,dc=example,dc=com")
 (targetattr = "*")
 (version 3.0; acl "Allow self management of profiles";
 allow (delete, write)(userdn = "ldap:///self");)

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

```
ou: People
cn: Barbara Jensen
cn: Babs Jensen
givenname: Barbara
sn: Jensen
userpassword: hifalutin
email: {"value": "bjensen@example.com", "type": "work", "primary": true}
phoneNumber: {"value": "+1 408 555 1862", "type": "work"}
ds-privilege-name: password-reset
$ import-ldif \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backendID dsEvaluation \
 --ldifFile profiles.ldif \
 --trustAll
```

6. (Optional) To check your work, read Babs's profile over LDAP:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDn dc=example,dc=com \
 "(uid=bjensen)"
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: profileUser
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
cn: Barbara Jensen
cn: Babs Jensen
email: {"value": "bjensen@example.com", "type": "work", "primary": true}
givenName: Barbara
ou: People
phoneNumber: {"value": "+1 408 555 1862", "type": "work"}
sn: Jensen
uid: bjensen
```

Now that you have sample data in LDAP, create the REST to LDAP mapping.

Writing the REST to LDAP Mapping

The REST to LDAP mapping defines the HTTP API and JSON resources that are backed by the LDAP service and entries.

Rather than patch the default example mapping, this example adds a separate API with a new mapping for the profile view.

The new mapping uses the following features of interest:

- A `"serverNaming"` naming strategy ensures proper handing of Common REST `"_id"` values. LDAP entries use `uid` as the RDN. JSON profiles use server-generated `"_id"` values.

The JSON profiles have `"externalId"` values corresponding to the LDAP `uid`. However, the JSON profiles do not define `"_id"` values. Client applications will use the `"_id"` values when uniquely identifying a profile, but not when looking up a profile.

The `"_id"` maps to the `entryUUID` LDAP operational attribute. The `entryUUID` attribute is generated on creation and managed by the LDAP server.

- As in the default example, passwords are not visible in profiles.

  Client applications use dedicated REST actions to manage password reset by administrators and to manage password modifications by users. The REST actions for password management require HTTPS.

- The `"externalId"` and `"userName"` fields both map to LDAP `uid` attributes.

  You could make it possible to update these fields after creation. This example assumes that these fields might be expected not to change, and so restricts client applications from changing them.

- The `"formatted"` and `"familyName"` fields of the name map to attributes that are required by the LDAP object class, `inetOrgPerson`. They are therefore required in the JSON profile as well.

The full mapping file is shown in the following listing:

```
{
    "version": "1.0",
    "resourceTypes": {
        "example-mapping": {
            "subResources": {
                "users": {
                    "type": "collection",
                    "dnTemplate": "ou=people,dc=example,dc=com",
                    "resource": "examples:user:1.0",
                    "namingStrategy": {
                        "type": "serverNaming",
                        "dnAttribute": "uid",
                        "idAttribute": "entryUUID"
                    }
                }
            }
        },
        "frapi:opendj:rest2ldap:object:1.0": {
            "isAbstract": true,
            "objectClasses": [
                "top"
            ],
            "resourceTypeProperty": "_schema",
            "properties": {
                "_schema": {
                    "type": "resourceType"
                },
                "_rev": {
                    "type": "simple",
                    "ldapAttribute": "etag",
                    "writability": "readOnly"
                },
```

```
                "_meta": {
                    "type": "object",
                    "properties": {
                        "created": {
                            "type": "simple",
                            "ldapAttribute": "createTimestamp",
                            "writability": "readOnly"
                        },
                        "lastModified": {
                            "type": "simple",
                            "ldapAttribute": "modifyTimestamp",
                            "writability": "readOnly"
                        }
                    }
                }
            }
        },
        "examples:user:1.0": {
            "superType": "frapi:opendj:rest2ldap:object:1.0",
            "objectClasses": [
                "profileUser",
                "person",
                "organizationalPerson",
                "inetOrgPerson"
            ],
            "supportedActions": [
                "modifyPassword",
                "resetPassword"
            ],
            "properties": {
                "externalId": {
                    "type": "simple",
                    "ldapAttribute": "uid",
                    "isRequired": true,
                    "writability": "createOnlyDiscardWrites"
                },
                "userName": {
                    "type": "simple",
                    "ldapAttribute": "uid",
                    "isRequired": true,
                    "writability": "createOnlyDiscardWrites"
                },
                "name": {
                    "type": "object",
                    "properties": {
                        "formatted": {
                            "type": "simple",
                            "ldapAttribute": "cn",
                            "isRequired": true
                        },
                        "givenName": {
                            "type": "simple"
                        },
                        "familyName": {
                            "type": "simple",
                            "ldapAttribute": "sn",
                            "isRequired": true
                        }
                    }
```

```
                },
                "emails": {
                    "type": "json",
                    "ldapAttribute": "email",
                    "isMultiValued": true
                },
                "phoneNumbers": {
                    "type": "json",
                    "ldapAttribute": "phoneNumber",
                    "isMultiValued": true
                }
            }
        }
    }
}
```

For details on all mapping features, read "Mapping Configuration File" in the *Reference*.

To Implement the REST to LDAP Mapping

Follow these steps to configure and activate the REST API based on the mapping:

1. Download the mapping file, .
2. Copy the mapping file to the appropriate server REST to LDAP configuration directory:

   ```
   $ mkdir /path/to/opendj/config/rest2ldap/endpoints/rest
   $ cp example-mapping.json /path/to/opendj/config/rest2ldap/endpoints/rest/
   ```

   As the default example defines a REST API under `/api`, this example uses `/rest` instead.
3. Enable a Rest2ldap endpoint for the API:

   ```
   $ dsconfig \
    create-http-endpoint \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --set authorization-mechanism:HTTP\ Basic \
    --set config-directory:config/rest2ldap/endpoints/rest \
    --set enabled:true \
    --type rest2ldap-endpoint \
    --endpoint-name /rest \
    --trustAll \
    --no-prompt
   ```

Using the REST API

This section demonstrates how client applications can use the REST API that you have defined.

Looking Up a Profile by Email Address

The following query looks for profiles with email addresses containing `bjensen`:

```
$ curl \
 --request GET \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/rest/users/?_queryFilter=emails/value+co+'bjensen'"
{
  "result": [
    {
      "_id": "<id>",
      "_rev": "<rev>",
      "_schema": "examples:user:1.0",
      "_meta": {},
      "externalId": "bjensen",
      "userName": "bjensen",
      "name": {
        "formatted": [
          "Barbara Jensen",
          "Babs Jensen"
        ],
        "givenName": "Barbara",
        "familyName": "Jensen"
      },
      "emails": [
        {
          "value": "bjensen@example.com",
          "type": "work",
          "primary": true
        }
      ],
      "phoneNumbers": [
        {
          "value": "+1 408 555 1862",
          "type": "work"
        }
      ]
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Looking Up a Profile by Phone Number

The following query looks for profiles with the phone number `+1 408 555 1862`:

```
$ curl \
 --request GET \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/rest/users/?_queryFilter=phoneNumbers/value+eq
+'%2B1%20408%20555%201862'"
```

Notice that the telephone number is URL encoded as `%2B1%20408%20555%201862`.

The output is the same as for the previous example.

Creating a User Profile

This example creates a profile using the JSON in .

```
$ curl \
 --request POST \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data @newuser.json \
 http://opendj.example.com:8080/rest/users/
```

Upon initial creation, the user has no password, and so cannot authenticate, yet.

Setting a Password

In order to allow the user to authenticate, the following example sets a password chosen by the user, which in this case is `password`.

This example uses Bash shell and jq to manipulate JSON on the command line.

The first operation gets the user profile `"_id"` and holds it in an `ID` environment variable.

The second operation performs an administrative password reset as Babs and holds the resulting generated password in a `PASSWORD` environment variable.

The third and final operation uses the generated password to authenticate as the user and modify the password:

```
$ export ID=$(jq --raw-output '.result[0] ._id' \
 <(curl --request GET --user bjensen:hifalutin 2>/dev/null \
 "http://opendj.example.com:8080/rest/users/?_queryFilter=externalId+eq+'newuser'"))
$ export PASSWORD=$(jq --raw-output '.generatedPassword' \
 <(curl --insecure --request POST --user bjensen:hifalutin \
 --header "Content-Type: application/json" --data '{}' \
 "https://opendj.example.com:8443/rest/users/${ID}?_action=resetPassword"))
$ curl \
 --insecure \
 --request POST \
 --user "newuser:${PASSWORD}" \
 --header "Content-Type: application/json" \
 --data "{\"oldPassword\": \"${PASSWORD}\", \"newPassword\": \"password\"}" \
 "https://opendj.example.com:8443/rest/users/${ID}?_action=modifyPassword"
```

This example is not meant to be secure, but instead demonstrates the process without security. In a production deployment, make sure the client can trust the server certificate for the HTTPS connection. Also, do not store passwords in clear text in the environment.

Reading a User's Own Profile

This example employs the user profile `"_id"` obtained as shown above:

```
$ curl \
 --request GET \
 --user newuser:password \
 "http://opendj.example.com:8080/rest/users/${ID}"
```

Changing a User's Own User-Defined Email Type

This example employs the user profile `"_id"` obtained as shown above.

Download the JSON patch payload, :

```
$ curl \
 --request PATCH \
 --user newuser:password \
 --header "Content-Type: application/json" \
 --data @changeEmailType.json \
 "http://opendj.example.com:8080/rest/users/${ID}"
{
  "_id": "<id>",
  "_rev": "<rev>",
  "_schema": "examples:user:1.0",
  "_meta": {
    "created": "<date>",
    "lastModified": "<date>"
  },
  "externalId": "newuser",
  "userName": "newuser",
  "name": {
    "formatted": "New User",
    "givenName": "User",
    "familyName": "New"
  },
  "emails": [
    {
      "value": "newuser@example.com",
      "type": "work",
      "primary": true
    },
    {
      "value": "newuser@example.org",
      "type": "home",
      "primary": false
    }
  ],
  "phoneNumbers": [
    {
      "value": "+1 408 555 1212",
      "type": "work"
    }
  ]
}
```

Removing a User's Own Email Address

This example employs the user profile `"_id"` obtained as shown above.

Download the JSON patch payload, :

```
$ curl \
 --request PATCH \
 --user newuser:password \
 --header "Content-Type: application/json" \
```

```
 --data @removeWorkEmail.json \
 "http://opendj.example.com:8080/rest/users/${ID}"
{
  "_id": "<id>",
  "_rev": "<rev>",
  "_schema": "examples:user:1.0",
  "_meta": {
    "created": "<date>",
    "lastModified": "<date>"
  },
  "externalId": "newuser",
  "userName": "newuser",
  "name": {
    "formatted": "New User",
    "givenName": "User",
    "familyName": "New"
  },
  "emails": [
    {
      "value": "newuser@example.org",
      "type": "home",
      "primary": true
    }
  ],
  "phoneNumbers": [
    {
      "value": "+1 408 555 1212",
      "type": "work"
    }
  ]
}
```

Adding a User's Own Cell Phone Number

This example employs the user profile `"_id"` obtained as shown above.

Download the JSON patch payload, :

```
$ curl \
 --request PATCH \
 --user newuser:password \
 --header "Content-Type: application/json" \
 --data @addPersonalCell.json \
 "http://opendj.example.com:8080/rest/users/${ID}"
{
  "_id": "32f27c44-6fac-4584-acad-748df4aa25d5",
  "_rev": "000000004c17d8fc",
  "_schema": "examples:user:1.0",
  "_meta": {
    "created": "2018-07-05T14:25:05Z",
    "lastModified": "2018-07-05T14:29:40Z"
  },
  "externalId": "newuser",
  "userName": "newuser",
  "name": {
    "formatted": "New User",
    "givenName": "User",
    "familyName": "New"
```

```
  },
  "emails": [
    {
      "value": "newuser@example.org",
      "type": "home",
      "primary": true
    }
  ],
  "phoneNumbers": [
    {
      "value": "+1 408 555 1212",
      "type": "work"
    },
    {
      "value": "+1 408 555 1234",
      "type": "personal cell"
    }
  ]
}
```

Deleting a User's Own Profile

This example employs the user profile `"_id"` obtained as shown above:

```
$ curl \
 --request DELETE \
 --user newuser:password \
 --header "Content-Type: application/json" \
 "http://opendj.example.com:8080/rest/users/${ID}"
```

Working With REST API Documentation

As described in "Common REST API Documentation" in the *Developer's Guide*, API descriptors provide runtime documentation for REST APIs. Although it is possible to serve the descriptors at runtime, do not use production servers for this purpose. Instead, prepare the documentation by reading API descriptors from a server with the same API as production servers, and publish the documentation separately.

This section includes the following procedures:

- "To Prepare Final API Documentation for Rest2ldap Endpoints"
- "To Protect Production Servers"

To Prepare Final API Documentation for Rest2ldap Endpoints

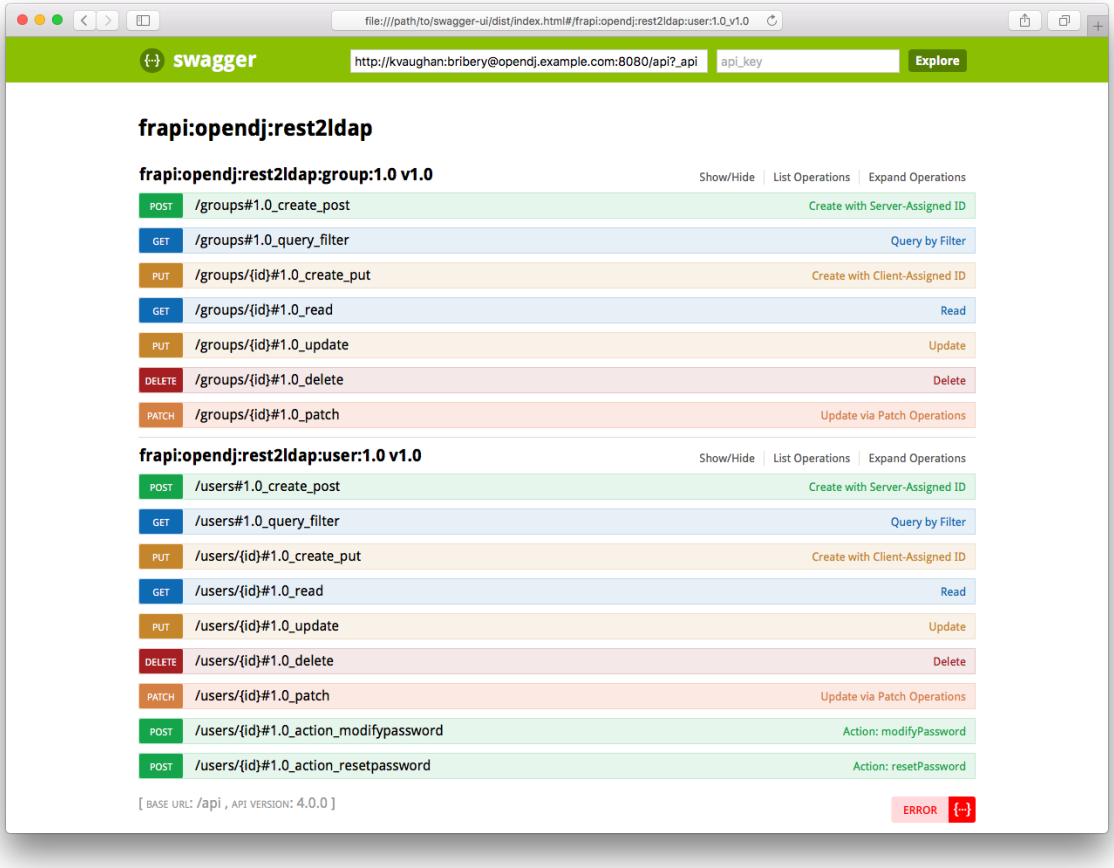
Preparing documentation for a Rest2ldap endpoint is an iterative process.

1. Configure the API as described in "Mapping Configuration File" in the *Reference*.
2. Run a local copy of a tool for viewing OpenAPI documentation, such as Swagger UI.
3. View the generated documentation through the tool by reading the OpenAPI format descriptor.

For example, read the descriptor for the `/api` endpoint with a URL such as `http://kvaughan:bribery@opendj.example.com:8080/api?_api`.

"Generated API Documentation" shows the documentation as it appears using the Swagger UI browser tool.

Generated API Documentation



If your browser does not display the generated documentation, turn off CORS settings in your browser. See your browser's documentation or search the web for details.

4. Update the API configuration as necessary.
5. Force the Rest2ldap endpoint to reread the updated configuration file.

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

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

6. Edit the descriptor as suggested in "To Publish OpenAPI Documentation" in the *Developer's Guide*.

7. Publish the final descriptor alongside your production deployment.

To Protect Production Servers

Protect production servers from unwanted API descriptor requests:

- What you do depends on how you deploy REST to LDAP functionality.
  - If you allow direct access through an HTTP connection handler, follow these steps:
    a. Set `api-descriptor-enabled` to `false`:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name HTTP \
 --set api-descriptor-enabled:false \
 --trustAll \
 --no-prompt
```

    b. Restart the connection handler:

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

- If you use the REST to LDAP gateway, follow these steps:

  a. Identify the REST to LDAP endpoints you expose.

  b. In the configuration of the HTTP service where the gateway runs, prevent requests to the API descriptors on those endpoints.

     Requests for API descriptors are requests that use the reserved query string parameters, `_api` and `_crestapi`.

     If you expect the requests to come from client applications, you could have the HTTP gateway respond immediately with HTTP status code `501 Not Implemented`, or `404 Not Found`, and stop processing the request.

     If you expect the requests to come from users, you could have the HTTP gateway respond immediately with HTTP status code `301 Moved Permanently` and a `Location` header to redirect the user-agent to the URL of the documentation you publish, and stop processing the request.

Chapter 16

Configuring LDAP Proxy Services

This chapter shows how to configure a server to forward LDAP requests to remote directory servers. For instructions on setting up a server as a proxy server, see "*Installing a Directory Proxy Server*" in the *Installation Guide* instead. In this chapter you will learn how to:

- Configure a discovery mechanism for remote directory servers
- Configure routing to remote directory servers
- Choose an appropriate load balancing algorithm
- Align schema definitions between a proxy and remote directory servers
- Configure a proxy backend that forwards LDAP requests
- Use access controls and resource limits with proxy services
- Use proxy services to deploy a highly available directory service
- Use proxy services to provide a single point of access to a directory service

About Proxy Services

Directory Services proxy services enable you to build a single point of access to a directory service, with a uniform view of the underlying LDAPv3 user data that hides implementation details from directory client applications, and promotes scalability and availability.

You can set up a server as a directory proxy server. For details, see "*Installing a Directory Proxy Server*" in the *Installation Guide*. When you set up a directory proxy server, no user data is stored locally. The server acts purely as an LDAP proxy. The only local data is for the server configuration and LDAP schema. In addition, the server is set up to use global access control policies rather than global ACIs. Global access control policies provide coarse-grained access control suitable for use on proxy servers, where the lack of local access to directory data makes ACIs a poor fit. For details, see "About Global Access Control Policies".

Note

Access control instructions (ACI) and global access control policies rely on different access control handlers, which implement different access control models. ACIs rely on the DSEE-compatible access control handler.

(DSEE refers to Sun Java System Directory Server Enterprise Edition.) Global access control policies rely on the policy-based access control handler. A server can only use one handler at a time.

Take the following constraints into consideration:

- When the policy-based handler is configured, ACIs have no effect.
- When the DSEE-compatible handler is configured, global access control policies have no effect.
- When a server is set up as a directory server, it uses the DSEE-compatible access control handler, with ACIs in directory data and global ACIs in the configuration.
- When a server is set up as a directory proxy server, it uses the policy-based access control policy handler, and global access control policies.
- Once the server has been set up, the choice of access control handler cannot be changed with the **dsconfig** command.

The rest of this chapter describes how to configure proxy services when the server has already been set up.

Proxy services are provided by proxy backends. Proxy backends connect to remote directory servers using a dynamic and configurable discovery mechanism as described in "Connecting to Remote Directory Servers". They route requests to remote directory servers as described in "Routing Requests to Remote Directory Servers". The way they distribute requests is configurable as described in "Choosing Load Balancing Settings". The way they handle failures when processing forwarded requests is described in "Understanding How Failures Are Handled".

LDAP schema definitions for user data must be aligned on the proxy server and on the remote directory servers. For more information, see "Managing Schema Definitions".

ACIs are handled by the directory server where the target data is stored. In other words, global access control policies set on a proxy server do not change ACIs on the remote directory server. Set ACIs appropriately on the directory server independently of proxy settings.

You can set up a proxy backend as described in "Configuring a Proxy Backend". If you require only one proxy backend, set up the server as a directory proxy server. For details, see "*Installing a Directory Proxy Server*" in the *Installation Guide*. For additional deployment suggestions, see "Deploying Proxy Services for High Availability" and "Deploying a Single Point of Directory Access".

Connecting to Remote Directory Servers

When the target DN of an LDAP request is not in a local backend, an LDAP server can refuse to handle the request, or return a referral. An LDAP proxy can also forward the request to another directory server.

In Directory Services, the LDAP proxy is implemented as a proxy backend. Rather than store user data locally, the proxy backend forwards requests to remote directory servers.

For proxy backends, a *service discovery mechanism* identifies remote directory servers to forward LDAP requests to. A service discovery mechanism's configuration specifies the keys used for secure communications, and how to contact the remote directory servers. It reads remote directory servers' configurations to discover their capabilities, including the naming contexts they serve, and so which target DNs they can handle. It periodically rereads their configurations in case they have been updated since the last service discovery operation.

When preparing to configure a service discovery mechanism, choose one of these alternatives:

Replication service discovery mechanism

This mechanism contacts DS replication servers to discover directory servers to forward LDAP requests to. Each replication server maintains information about the replication topology that allows the proxy server to discover directory server replicas.

This mechanism only works with replicated DS servers.

A replication service discovery mechanism configuration includes a bind DN and password to connect to replication servers. It uses this account to read configuration data under `cn=admin data` and `cn=config`. The account must have access and privileges to read that configuration data, and it must exist with the same credentials on all replication servers.

Static service discovery mechanism

This mechanism maintains a static list of directory server *`host:port`* combinations. You must enumerate the servers to forward LDAP requests to.

This mechanism is designed to work with all LDAPv3 directory servers that support proxied authorization.

When configuring a service discovery mechanism, make sure that all the remote directory servers are replicas of each other, and that they have the same capabilities. A proxy backend expects all remote directory server replicas known to the mechanism to hold the same data. This allows the backend to treat the replicas as equivalent members of a pool. In the configuration, a pool of equivalent replicas is a *shard*.

In deployments where you must distribute data for horizontal write scalability, you can configure multiple service discovery mechanisms. The proxy backend can then distribute write requests across multiple shards, as described in "Routing Requests to Remote Directory Servers".

The following example creates a replication service discovery mechanism that specifies two replication servers to contact securely:

```
$ dsconfig \
 create-service-discovery-mechanism \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mechanism-name "Replication Service Discovery Mechanism" \
 --type replication \
 --set bind-dn:"cn=admin,cn=Administrators,cn=admin data" \
 --set bind-password:password \
 --set replication-server:rs1.example.com:4444 \
 --set replication-server:rs2.example.com:4444 \
 --set use-ssl:true \
 --set trust-manager-provider:"JVM Trust Manager" \
 --trustAll \
 --no-prompt
```

The following example creates a static service discovery mechanism that specifies four directory servers to contact:

```
$ dsconfig \
 create-service-discovery-mechanism \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mechanism-name "Static Service Discovery Mechanism" \
 --type static \
 --set primary-server:local1.example.com:636 \
 --set primary-server:local2.example.com:636 \
 --set secondary-server:remote1.example.com:636 \
 --set secondary-server:remote2.example.com:636 \
 --set use-ssl:true \
 --set trust-manager-provider:"JVM Trust Manager" \
 --trustAll \
 --no-prompt
```

The examples above assume that the remote servers use certificates signed by well-known CAs, and so are recognized using the trust manager provided by the JVM. If this is not the case, configure an appropriate trust manager provider.

If the proxy server must perform SSL mutual authentication when setting up secure connections with the remote servers, configure an appropriate key manager provider and SSL certificate nickname.

Supported service discovery mechanisms define a `discovery-interval` that specifies how often to read the remote server configurations in order to discover changes. Because the mechanism polls periodically for configuration changes, by default, it can take up to one minute for the mechanism to see the changes. If necessary, you can change this setting in the configuration.

Routing Requests to Remote Directory Servers

A proxy backend forwards requests according to their target DNs. The proxy backend matches target DNs to base DNs that you specify in the configuration. In addition, if you specify multiple shards for data distribution, the proxy also forwards requests to the appropriate shard.

When specifying base DNs, bear in mind the following points:

- A server responds first with local data, such as the server's own configuration or monitoring data. If a request target DN cannot be served from local data, the proxy backend can forward the request to a remote directory server.

  The proxy backend will forward the request if its target DN is under one of the specified base DNs.

- As an alternative to enumerating a list of base DNs to proxy, you can set the proxy backend property `route-all: true`.

  When you activate this property, the proxy backend will attempt to forward all requests that can be served by public naming contexts of remote servers. If the request target DN is not in a naming context supported by any remote directory servers associated with the proxy backend, then the proxy will not forward the request.

- The LDAP proxy capability is intended to proxy user data, not server-specific data.

  A server with a proxy backend still responds directly to requests that target private naming contexts such as `cn=config`, `cn=tasks`, and `cn=monitor`. Local backends hold configuration and monitoring information that is specific to the server, and these naming contexts are not public.

  Make sure that client applications for configuration and monitoring access servers directly for the server-specific data they need.

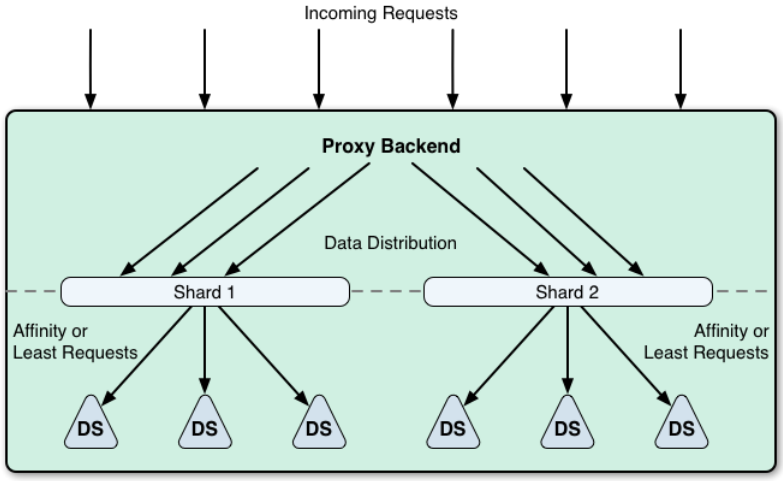
- If you configure multiple proxy backends, each proxy backend must target distinct base DNs.

In deployments where you must distribute data for horizontal write scalability, you can configure multiple service discovery mechanisms for the same base DNs. Each service discovery mechanism targets its own distinct shard of directory data.

To enable write scalability beyond what is possible with a single shard of replicas, each shard must have its own independent replication configuration. Replication replays each update only within the shard.

The proxy backend distributes write requests across the shard, and balances the load within each shard. This is shown in "How a Proxy Backend Routes Requests".

How a Proxy Backend Routes Requests



Data distribution for horizontal scalability has the following properties:

- The proxy backend always routes requests targeting an entry below the partition base DN to the *same* shard.
- The proxy backend routes read requests targeting the partition base DN entry or above to *any* shard.

  In other words, the proxy backend can route each request to a different shard. When you deploy data distribution, the proxy rejects writes to the partition base DN entry and above through the proxy backend. Instead, you must perform such write operations on a replica in each shard.

  The best practice is therefore to update the partition base DN entry and above prior to deployment.
- The proxy backend routes search requests to *all* shards unless the search scope is below the partition base DN.

  For example, if the partition base DN is `ou=people,dc=example,dc=com`, the proxy backend always routes a search with base DN `uid=bjensen,ou=people,dc=example,dc=com` or `deviceid=12345,uid=bjensen,ou=people,dc=example,dc=com` to the same shard. However, it routes a search with base DN `ou=people,dc=example,dc=com` to all shards.

For an example, see "Scaling Out Using Data Distribution" in the *Deployment Guide*.

Choosing Load Balancing Settings

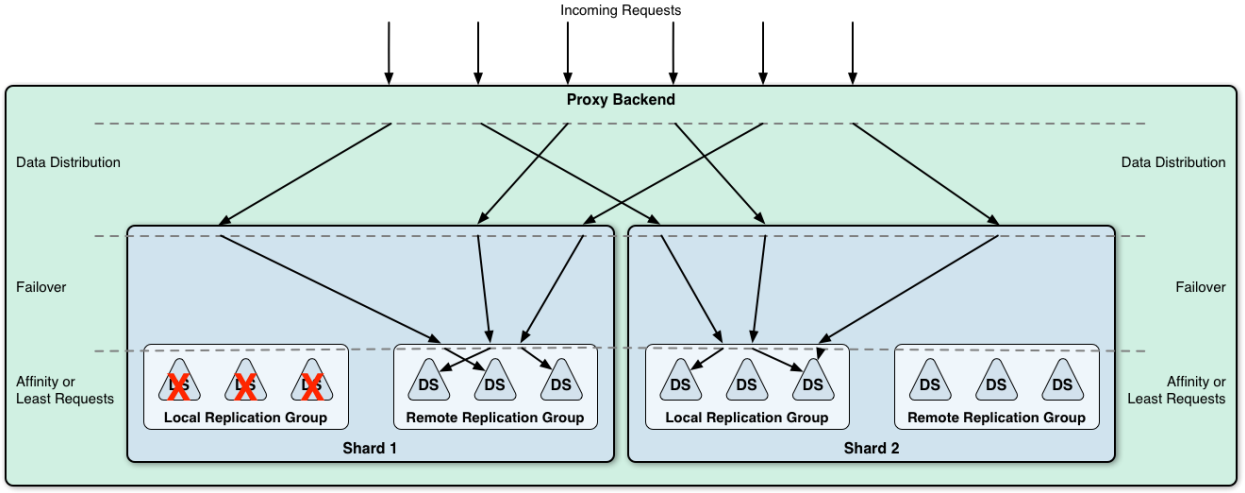
A directory proxy server balances the LDAP request load across the remote directory servers it forwards requests to.

The proxy performs load balancing for the following purposes:

- **Data distribution**: sharding data to scale out
- **Failover**: routing requests to available directory servers
- **Affinity**: consistently forwarding requests for the same entry to the same server
- **Least requests**: equitably routing requests based on each directory server's load

The proxy applies these load balancing capabilities in hierarchical order to route requests. Follow the paths of requests through the proxy backend to a directory server in this diagram:

Load Balancing in Directory Proxy Servers



Notice in the diagram how the load balancing capabilities work together to route a request to a directory server:

- Data distribution routes to the correct shard.
- Failover routes around directory servers that are down, routing the request to available servers.
- Affinity or least requests routes the operation to a specific server.

Read the following to determine which settings to use:

Data Distribution

Description

Routes requests with the same target DN below the partition base DN to the same shard. Data distribution helps to scale out the directory service horizontally.

When to Use

Use when you must scale out the directory service, and the deployment fits the constraints described in "Scaling Out Using Data Distribution" in the *Deployment Guide*.

A single DS directory service shard can process thousands of LDAP requests *per second* given the proper configuration and tuning. Furthermore, you can increase the search and read throughput simply by adding replicas. Configure data distribution for deployments when performance testing demonstrates that you cannot achieve write throughput requirements with properly configured and tuned replicas in a single replication topology.

For example, use data distribution for a very high scale AM CTS deployment.

Settings

See "Scaling Out Using Data Distribution" in the *Deployment Guide*.

Failover

Description

Routes LDAP requests to LDAP servers that are available and responding to health check probe requests. Failover prevents the proxy server from continuing to forward requests to unavailable servers.

Also see "Understanding How Failures Are Handled".

When to Use

Use failover settings to route requests to local servers if they are available, and remote servers only when local servers are not available.

Settings

Proxy backend properties:

`connection-timeout`
`heartbeat-interval`
`heartbeat-search-request-base-dn`

Replication service discovery mechanism property:

`primary-group-id`

Static service discovery mechanism properties:

`primary-server`
`secondary-server`

Affinity

Description

Routes LDAP requests with the same target DN to the same server.

Affinity load balancing helps applications that update and then reread the same entry in quick succession. This is not a best practice, but is often seen in client applications.

With an add or modify request on an entry that is quickly followed by a read of the entry, the requests to replicate the update can take longer than the read request, depending on network latency. Affinity load balancing forwards the read request to the same server that processed the update, ensuring that the client application obtains the expected result.

Affinity routing depends on the values of the proxy backend property, `partition-base-dn`. The proxy consistently routes requests for entries subordinate to these entries to the same server. The values of this property should therefore be the lowest entries in your DIT that are part of the DIT structure and not part of application data. In other words, when using affinity with two main branches, `ou=groups,dc=example,dc=com` and `ou=people,dc=example,dc=com`, set:

```
partition-base-dn:ou=groups,dc=example,dc=com
partition-base-dn:ou=people,dc=example,dc=com
```

In terms of the CAP theorem, affinity load balancing provides consistency and availability, but not partition tolerance. As this algorithm lacks partition tolerance, configure it to load balance requests in environments where partitions are unlikely, such as a single data center with all directory servers on the same network.

When to Use

Use whenever possible, in combination with failover. Client application developers may be unaware of LDAP best practices.

Settings

Proxy backend properties:

`load-balancing-algorithm`
`partition-base-dn`

Least Requests

Description

Routes requests to the LDAP directory server with the least requests currently being serviced.

Least requests load balancing helps to spread requests equitably across a pool of replicated servers.

In terms of the CAP theorem, least requests load balancing provides availability and partition tolerance, but not consistency. A write request followed by a read request of the same entry can be routed to a different directory server.

When to Use

Use only when you cannot use affinity instead.

Settings

Proxy backend property:

`load-balancing-algorithm`

Managing Schema Definitions

Proxy services are designed to proxy user data rather than server configuration or monitoring data. In addition, a proxy server must expose the same LDAP schema for user data as the remote directory servers.

If the schema definitions for user data differ between the proxy server and the remote directory servers, you must update them to bring them into alignment.

For details on DS server schema, see "*Managing Schema*".

If the remote servers are not DS servers, also see the schema documentation for the remote servers. Schema formats are not identical across all LDAPv3 servers. You will need to translate the differences into native formats, and apply changes locally on each server.

Configuring a Proxy Backend

This section shows how to create a proxy backend once you have made your configuration choices. As described in "About Proxy Services", how the server was set up determines the access control model. The access control model for proxy servers is mutually exclusive with the model for directory servers.

Create proxy backends only on servers set up as proxy servers (with **setup proxy-server**).

Important

A directory proxy server connects using an account that must exist in the remote directory service.

The directory proxy server binds with this account, and then forwards LDAP requests on behalf of other users.

The proxy account must have the following on all remote directory servers:

- The same bind credentials, such as bind DN and bind password
- The right to perform proxied authorization

Examples in the documentation use the account with bind DN `cn=Proxy,ou=Apps,dc=example,dc=com` and bind password `password`:

```
dn: cn=Proxy,ou=Apps,dc=example,dc=com
cn: Proxy
objectClass: top
objectClass: applicationProcess
objectClass: simpleSecurityObject
userPassword: password
ds-privilege-name: proxied-auth
```

The following example ACI on `dc=example,dc=com` grants applications the rights to perform proxied authorization:

```
aci: (target="ldap:///dc=example,dc=com") (targetattr ="*")
  (version 3.0; acl "Allow apps proxied auth"; allow(all, proxy)
  (userdn = "ldap:///cn=*,ou=Apps,dc=example,dc=com");)
```

To grant the rights to the proxy account alone, set `userdn = ldap:///cn=Proxy,ou=Apps,dc=example,dc=com` instead.

When using DS directory services, also see "Configuring Proxied Authorization" in the *Developer's Guide* for details. Otherwise, read about how to set up proxied authorization in your directory server documentation.

This section includes the following examples:

- "Proxy: Forward Requests for Example.com"
- "Proxy: Forward All User Data Requests"

Proxy: Forward Requests for Example.com

The following example creates a proxy backend that forwards LDAP requests when the target DN is in `dc=example,dc=com`. It uses the replication service discovery mechanism described in "Connecting to Remote Directory Servers", and affinity load balancing described in "Choosing Load Balancing Settings":

```
$ dsconfig \
 create-backend \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name proxyExampleCom \
 --type proxy \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --set route-all:false \
 --set proxy-user-dn:cn=proxy,ou=apps,dc=example,dc=com \
 --set proxy-user-password:password \
 --set load-balancing-algorithm:affinity \
 --set partition-base-dn:ou=groups,dc=example,dc=com \
 --set partition-base-dn:ou=people,dc=example,dc=com \
 --set shard:"Replication Service Discovery Mechanism" \
 --set heartbeat-search-request-base-dn:cn=proxy,ou=apps,dc=example,dc=com \
 --trustAll \
 --no-prompt
```

This command fails if `dc=example,dc=com` is already served by local data.

Notice the setting for `heartbeat-search-request-base-dn`. By default, the proxy backend sends periodic heartbeat requests to remote directory servers to help manage its connections. "Understanding How Failures Are Handled" describes in more detail how a proxy backend uses heartbeats.

By default, heartbeat requests target the remote directory server root DSE. A response from the root DSE suffices to determine whether the server is up, but it does not indicate whether the proxy can access data under a given base DN. Set `heartbeat-search-request-base-dn` to target an entry located under a base DN of interest.

Proxy: Forward All User Data Requests

The following example creates a proxy backend that forwards LDAP requests targeting user data, not specifying base DNs explicity, but instead using the `route-all` property. It uses the static service discovery mechanism described in "Connecting to Remote Directory Servers", and least requests load balancing described in "Choosing Load Balancing Settings":

```
$ dsconfig \
 create-backend \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name proxyAll \
 --type proxy \
 --set enabled:true \
 --set route-all:true \
 --set proxy-user-dn:cn=proxy,ou=apps,dc=example,dc=com \
 --set proxy-user-password:password \
 --set load-balancing-algorithm:least-requests \
 --set shard:"Static Service Discovery Mechanism" \
 --trustAll \
 --no-prompt
```

The examples above do not specify how to secure proxy connections to remote directory servers. Connection security is defined using configuration properties of the service discovery mechanism that the proxy backend depends on.

Proxy backends define a `discovery-interval` that specifies how often to read the remote server configurations in order to discover changes. Because the proxy polls periodically for configuration changes, by default, it can take up to one minute to see the changes. If necessary, you can change this setting in the configuration.

Understanding How Failures Are Handled

As shown in "*LDAP Result Codes*" in the *Reference*, there are many specific ways that an LDAP request can fail. Not all failure result codes indicate a permanent failure, however. The following result codes from a remote directory server indicate a temporary failure:

- 51 (Busy) indicates that the server was too busy to process the request.

  The request can safely be tried on another peer server.

- 52 (Unavailable) indicates that the server was missing at least one resource needed to process the request.

  The request can safely be tried on another peer server.

When a forwarded request finishes with one of these return codes, the proxy backend retries the request on another server. In this case, the client does not receive the result from the remote directory server.

When a forwarded request finishes with a permanent server-side error return code, the proxy backend returns the result to the client application. In this case, the client must handle the error.

Connection failures can prevent the remote directory server from responding at all. The proxy backend handles connection failures differently, depending on whether it is inherently safe to replay the operation:

- For operations that read directory data, including search and compare requests, the proxy backend retries the request if a connection failure occurs.
- For operations that write directory data, including add, delete, modify, and modify DN requests, the proxy backend does not retry the request if a connection failure occurs.

  When the connection fails during a write operation, it is not possible to determine whether the change was applied or not. It is not safe, therefore, to replay the request on another server.

  The proxy returns an error to the client application.
- Connection failures during bind operations cause the proxy to retry the bind.

  In the unlucky event of repeated connection failures on successive bind attempts combined with a password policy configured to lock an account after a certain number of consecutive failures, it is possible that this behavior could lock an account.

A proxy backend protects against failed and stale connections with feedback from periodic heartbeat requests to remote directory servers.

A heartbeat request serves these purposes:

- Checks that the remote directory server is still available.

  If the request fails, the proxy closes the unresponsive connection and connects to another remote directory server.
- Determines whether an unresponsive server has recovered.

  When the remote directory server responds again to heartbeat requests, the proxy can begin forwarding client requests to it again.
- Keeps the connection alive, preventing it from appearing idle and being forcefully closed.

If necessary, you can configure how often heartbeat requests are sent using the proxy backend `heartbeat-interval` property.

Applying Access Control and Resource Limits for Proxy Services

When you deploy a pure directory proxy server, where the server has only proxy backends and configuration and no local user data, you limit the mechanisms you can use to control access and limit resource use. Such mechanisms cannot rely on ACIs in user data, resource limit settings on user

entries, or group membership to determine what the server should allow. They can depend only on the server configuration and on the properties of the incoming request.

Global access control policies provide an access control mechanism that is suitable for pure directory proxy servers. For details, see "About Global Access Control Policies".

Resource limits set in the server configuration provide a way to prevent directory clients from using an unfair share of system resources. For details, see "*Setting Resource Limits*", which covers different methods, including how to update a server's configuration.

Deploying Proxy Services for High Availability

Directory services are designed for basic availability. Directory data replication makes it possible to continue reading and writing directory data even when the network is partitioned, meaning that remote servers cannot effectively contact each other. This sort of availability assumes, however, that client applications can handle temporary interruptions.

Some client applications can be configured to connect to a set of directory servers. Some of these clients are able to retry requests when a server is unavailable. Others expect a single point of entry to the directory service, or cannot continue promptly when a directory server is unavailable.

Individual directory servers can become unavailable for various reasons:

- A network problem can make it impossible to reach the server.
- The server can be temporarily down for maintenance (backup, upgrade, and other purposes).
- The server can be removed from a replication topology and replaced with a different server.
- A crash can bring the server down temporarily for a restart or permanently for a replacement.

All of these interruptions must be managed on the client side. Some could require reconfiguration of each client application.

A directory proxy server provides high availability to client applications by hiding these implementation details from client applications. The proxy presents client applications with a uniform view of the remote directory servers. It periodically rereads the remote servers' configurations so that it can route requests correctly even when remote servers change. It regularly checks connections to remote servers so that it can route requests correctly even when some servers are unavailable.

In addition, directory proxy servers are well-suited for applications that need a single entry point to the directory service. For details, see "Deploying a Single Point of Directory Access".

Deploying a Single Point of Directory Access

Unlike directory servers with their potentially large sets of volatile user data, standalone directory proxy servers manage only their configuration state. Proxy servers can start faster and require less

disk and memory space. Each directory proxy server can effectively be a clone of every other, with a configuration that does not change after server startup. Each clone routes LDAP requests and handles problems in the same way.

When you configure all directory proxy servers as clones of each other, you have a number of identical directory access points. Each point provides the same view of the underlying directory service. The points differ from each other only by their connection coordinates (host, port, and potentially key material to establish secure connections).

To consolidate identical access points to a single access point, configure your network to use a virtual IP address for the proxy servers. You can restart or replace proxy servers at any time, in the worst case losing only the client connections that were established with the individual proxy server.

An alternative to a single, global access point for all applications is to repeat the same approach for each key application. The proxy server configuration is specific to each key application. Clones with that configuration provide a single directory access point for the key application. Other clones do the same other for other key applications. Be sure to provide a more generic access point for additional applications.

Chapter 17

Configuring Pass-Through Authentication

This chapter focuses on pass-through authentication (PTA), whereby you configure another server to determine the response to an authentication request. A typical use case for PTA involves passing authentication through to Active Directory for users coming from Microsoft Windows systems. In this chapter you will learn to:

- Configure password policies to use PTA
- Assign PTA policies to users and to groups

About Pass-Through Authentication

You use *pass-through authentication* (PTA) when the credentials for authenticating are stored in a remote directory service instead of an DS service. In effect, the DS server redirects the bind operation against a remote LDAP server.

The method a server uses to redirect the bind depends on the mapping from the user entry in the DS server to the corresponding user entry in the remote directory. DS servers provide you several choices to set up the mapping:

- When both the local entry in the DS server and the remote entry in the other server have the same DN, you do not have to set up the mapping. By default, the DS server redirects the bind with the original DN and password from the client application.
- When the local entry in the DS server has been provisioned with an attribute holding the DN of the remote entry, you can specify which attribute holds the DN, and the DS server redirects the bind on the remote server using the DN value.
- When you cannot get the remote bind DN directly, you need an attribute and value on the DS entry that corresponds to an identical attribute and value on the remote server. In this case, you also need the bind credentials for a user who can search for the entry on the remote server. The DS server performs a search for the entry using the matching attribute and value, and then redirects the bind with the DN from the remote entry.

You configure PTA as an authentication policy that you associate with a user's entry in the same way that you associate a password policy with a user's entry. Either a user has an authentication policy for PTA, or the user has a local password policy.

Setting Up Pass-Through Authentication

When setting up pass-through authentication, you need to know to which remote server or servers to redirect binds, and you need to know how you map user entries in the DS server to user entries in the remote directory.

To Set Up SSL Communication For Testing

When performing PTA, you protect communications between the DS server and the server providing authentication. If you test secure connections with self-signed certificates, and you do not want the client to blindly trust the server, follow these steps to import the authentication server's certificate into the truststore used by the DS server.

1. Export the server certificate from the authentication server.

   How you perform this step depends on the authentication directory server. With the DS server, you can export the certificate as shown here:

   ```
   $ keytool \
    -exportcert \
    -rfc \
    -alias server-cert \
    -keystore /path/to/authn-server/config/keystore \
    -storepass:file /path/to/authn-server/config/keystore.pin \
    -storetype PKCS12 \
    -file authn-server-cert.pem
   ```

2. Record the host name used in the certificate.

   You use the host name when configuring the secure connection. With the DS server, you can view the certificate details as shown here:

   ```
   $ keytool \
    -list \
    -v \
    -alias server-cert \
    -keystore /path/to/authn-server/config/keystore \
    -storepass:file /path/to/authn-server/config/keystore.pin \
    -storetype PKCS12
   ...
   Owner: CN=authn-server.example.com, O=OpenDJ RSA Self-Signed Certificate
   Issuer: CN=authn-server.example.com, O=OpenDJ RSA Self-Signed Certificate
   ...
   ```

3. Import the authentication server certificate into the DS server's keystore:

```
$ keytool \
 -importcert \
 -alias authn-server-cert \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12 \
 -file authn-server-cert.pem \
 -noprompt
```

To Configure an LDAP Pass-Through Authentication Policy

You configure authentication policies with the **dsconfig** command. Notice that authentication policies are part of the server configuration, and therefore not replicated.

1. Set up an authentication policy for pass-through authentication to the authentication server:

```
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "PTA Policy" \
 --type ldap-pass-through \
 --set primary-remote-ldap-server:authn-server.example.com:1636 \
 --set mapped-attribute:uid \
 --set mapped-search-base-dn:"dc=example,dc=com" \
 --set mapping-policy:mapped-search \
 --set use-ssl:true \
 --trustAll \
 --no-prompt
```

   The policy shown here maps identities with this password policy to identities under `dc=example,dc=com` on the authentication server. Users must have the same `uid` values on both servers. The policy here also uses LDAPS between the DS server and the authentication server.

2. Check that your policy has been added to the list:

```
$ dsconfig \
 list-password-policies \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --property use-ssl \
 --trustAll \
 --no-prompt
Password Policy         : Type              : use-ssl
------------------------:-------------------:--------
Default Password Policy : password-policy   : -
PTA Policy              : ldap-pass-through : true
Root Password Policy    : password-policy   : -
```

To Configure Pass-Through Authentication To Active Directory

The steps below demonstrate how to set up PTA to Active Directory. Here is some information to help you make sense of the steps.

Entries on the DS side use `uid` as the naming attribute, and entries also have `cn` attributes. Active Directory entries use `cn` as the naming attribute. User entries on both sides share the same `cn` values. The mapping between entries therefore uses `cn`.

Consider the example where the DS account with `cn=LDAP PTA User` and DN `uid=ldapptauser,ou=People,dc=example,dc=com` corresponds to an Active Directory account with DN `CN=LDAP PTA User,CN=Users,DC=internal,DC=forgerock,DC=com`. The steps below enable the user with `cn=LDAP PTA User` on the DS server to authenticate through Active Directory:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 uid=ldapptauser \
 cn
dn: uid=ldapptauser,ou=People,dc=example,dc=com
cn: LDAP PTA User

$ ldapsearch \
 --hostname ad.example.com \
 --baseDN "CN=Users,DC=internal,DC=forgerock,DC=com" \
 --bindDN "cn=administrator,cn=Users,DC=internal,DC=forgerock,DC=com" \
 --bindPassword password \
 "(cn=LDAP PTA User)" \
 cn
dn: CN=LDAP PTA User,CN=Users,DC=internal,DC=forgerock,DC=com
cn: LDAP PTA User
```

The DS server must map its `uid=ldapptauser,ou=People,dc=example,dc=com` entry to the Active Directory entry, `CN=LDAP PTA User,CN=Users,DC=internal,DC=forgerock,DC=com`. In order to do the mapping, the DS

server has to perform a search for the user in Active Directory using the `cn` value that it recovers from its own entry for the user. Active Directory does not allow anonymous searches, so part of the authentication policy configuration consists of the administrator DN and password the DS server uses to bind to Active Directory to search.

Finally, before setting up the PTA policy, make sure the DS server can connect to Active Directory over a secure connection to avoid sending passwords in the clear.

1. Export the certificate from the Windows server.

   a. Click start > All Programs > Administrative Tools > Certification Authority, then right-click the CA and select Properties.

   b. In the General tab, select the certificate and click View Certificate.

   c. In the Certificate dialog, click the Details tab, then click Copy to File...

   d. Use the Certificate Export Wizard to export the certificate into a file, such as `windows.cer`.

2. Copy the exported certificate to the system running the DS server.

3. Import the server certificate into the DS keystore:

```
$ cd /path/to/opendj/config
$ keytool \
 -importcert \
 -alias ad-cert \
 -keystore keystore \
 -storepass:file keystore.pin \
 -storetype PKCS12 \
 -file ~/Downloads/windows.cer \
 -noprompt
Certificate was added to keystore
```

   At this point, the DS server can connect securely to Active Directory.

4. Set up an authentication policy for DS users to authenticate to Active Directory:

```
# Create a trust manager provider to access the Active Directory certificate:
$ dsconfig \
 create-trust-manager-provider \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name PKCS12 \
 --type file-based \
 --set enabled:true \
 --set trust-store-type:PKCS12 \
 --set trust-store-file:config/keystore \
 --set trust-store-pin:"&{file:config/keystore.pin}" \
 --trustAll \
```

```
 --no-prompt

# Use the trust manager provider in the pass-through authentication policy:
$ dsconfig \
 create-password-policy \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --type ldap-pass-through \
 --policy-name "AD PTA Policy" \
 --set primary-remote-ldap-server:ad.example.com:636 \
 --set mapped-attribute:cn \
 --set mapped-search-base-dn:"CN=Users,DC=internal,DC=forgerock,DC=com" \
 --set mapped-search-bind-dn:"cn=administrator,cn=Users,DC=internal, \
  DC=forgerock,DC=com" \
 --set mapped-search-bind-password:password \
 --set mapping-policy:mapped-search \
 --set trust-manager-provider:PKCS12 \
 --set use-ssl:true \
 --trustAll \
 --no-prompt
```

5. Assign the authentication policy to a test user:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: uid=ldapptauser,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=AD PTA Policy,cn=Password Policies,cn=config

Processing MODIFY request for uid=ldapptauser,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=ldapptauser,ou=People,dc=example,dc=com
```

6. Check that the user can bind using PTA to Active Directory:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --bindDN uid=ldapptauser,ou=People,dc=example,dc=com \
 --bindPassword password \
 "(cn=LDAP PTA User)" \
 userpassword cn
dn: uid=ldapptauser,ou=People,dc=example,dc=com
cn: LDAP PTA User
```

Notice that to complete the search, the user authenticated with a password to Active Directory, though no `userpassword` value is present on the entry on the DS side.

Assigning Pass-Through Authentication Policies

You assign authentication policies in the same way as you assign password policies, by using the `ds-pwp-password-policy-dn` attribute.

Note

Although you assign the pass-through authentication policy using the same attribute as for password policy, the authentication policy is not in fact a password policy. Therefore, the user with a pass-through authentication policy does not have a value for the operational attribute `pwdPolicySubentry`.

To Assign a Pass-Through Authentication Policy To a User

Users depending on PTA no longer need a local password policy, as they no longer authenticate locally.

Examples in the following procedure work for this user, whose entry in an DS directory server is as shown below. Notice that the user has no `userPassword` attribute. The user's password on the authentication server is `password`:

```
dn: uid=ptaUser,ou=People,dc=example,dc=com
uid: ptaUser
objectClass: top
objectClass: person
objectClass: organizationalperson
objectClass: inetorgperson
cn: PTA User
givenName: PTA
sn: User
homePhone: +1 225 216 5900
mail: ptaUser@example.com
```

This user's entry on the authentication server also has `uid=ptaUser`. The pass-through authentication policy performs the mapping to find the user entry in the authentication server.

1. Give an administrator access to update a user's password policy:

```
$ cat protect-pta.ldif
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "ds-pwp-password-policy-dn")
 (version 3.0;acl "Allow Kirsten Vaughan to assign password policies";
 allow (all) (userdn = "ldap:///uid=kvaughan,ou=People,dc=example,dc=com");)

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

2. Update the user's `ds-pwp-password-policy-dn` attribute:

```
$ cat pta-policy.ldif
dn: uid=ptaUser,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=PTA Policy,cn=Password Policies,cn=config

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

3. Check that the user can authenticate through to the authentication server:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --bindDN uid=ptaUser,ou=People,dc=example,dc=com \
 --bindPassword password \
 "(uid=ptaUser)" \
 1.1
dn: uid=ptaUser,ou=People,dc=example,dc=com
```

To Assign a Pass-Through Authentication Policy To a Group

Examples in the following steps use the PTA policy as defined above. The administrator's entry is present on the authentication server as well.

1. Give an administrator the privilege to write subentries, such as those used for password policies:

```
$ cat subentry-write.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: subentry-write

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

   Notice here that the directory superuser, `cn=Directory Manager`, assigns privileges to Kirsten Vaughan. Any administrator with the `privilege-change` privilege can assign other privileges. Assuming the administrator can update administrator privileges, they can assign themselves the `bypass-acl` privilege. Then they are no longer bound by access control instructions, including both user data ACIs and global ACIs. For this reason, avoid assigning the `privilege-change` privilege to normal administrator users.

2. Create a subentry for a collective attribute that sets the password policy `ds-pwp-password-policy-dn` attribute for group members' entries:

```
$ cat collective-pta.ldif
dn: cn=PTA Policy for Dir Admins,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: PTA Policy for Dir Admins
ds-pwp-password-policy-dn;collective: cn=PTA Policy,cn=Password Policies,cn=config
subtreeSpecification: { base "ou=People", specificationFilter
  "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)"}

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 collective-pta.ldif
```

   The `base` entry identifies the branch that holds administrator entries. For details on how subentries apply, see "Understanding Subentry Scope".

3. Check that the DS server has applied the policy.

   a. Make sure you can bind as the user on the authentication server:

```
$ ldapsearch \
 --hostname authn-server.example.com \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=People,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN "dc=example,dc=com" \
 "(uid=kvaughan)" \
 1.1
dn: uid=kvaughan,ou=People,dc=example,dc=com
```

   b. Check that the user can authenticate through to the authentication server from the DS server:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=kvaughan)" \
 1.1
dn: uid=kvaughan,ou=People,dc=example,dc=com
```

Chapter 18
Samba Password Synchronization

This chapter covers synchronization between directory passwords and Samba passwords. In this chapter you will learn to:

- Configure Samba for use with DS servers
- Set up the DS Samba password plugin for synchronization

Samba, the Windows interoperability suite for Linux and UNIX, stores accounts because UNIX and Windows password storage management is not interoperable. The default account storage mechanism is designed to work well with relatively small numbers of accounts and configurations with one domain controller. For larger installations, you can configure Samba to use DS directory servers for storing Samba accounts. See the Samba documentation for your platform for instructions on how to configure LDAP directory servers such as DS servers as Samba `passdb` backends.

The rest of this chapter focuses on how you keep passwords in sync when using your DS server for Samba account storage.

When you store Samba accounts in a directory server, Samba stores its own attributes as defined in the Samba schema. Samba does not use the LDAP standard `userPassword` attribute to store users' Samba passwords. You can configure Samba to apply changes to Samba passwords to LDAP passwords as well, too. Yet, if a user modifies their LDAP password directly without updating the Samba password, the LDAP and Samba passwords get out of sync.

The DS Samba Password plugin resolves this problem for you. The plugin intercepts password changes to Samba user profiles, synchronizing Samba password and LDAP password values. For an incoming Password Modify Extended Request or modify request changing the user password, the DS Samba Password plugin detects whether the user's entry reflects a Samba user profile (entry has object class `sambaSAMAccount`), hashes the incoming password value, and applies the password change to the appropriate password attribute, keeping the password values in sync. The DS Samba Password plugin can perform synchronization as long as new passwords values are provided in cleartext in the modification request. If you configure Samba to synchronize LDAP passwords when it changes Samba passwords, then the plugin can ignore changes by the Samba user to avoid duplicate synchronization.

To Set Up a Samba Administrator Account

The Samba Administrator synchronizes LDAP passwords after changing Samba passwords by issuing a Password Modify Extended Request. In Samba's `smb.conf` configuration file, the value of `ldap admin dn` is set to the DN of this account. When the Samba Administrator changes a user password, the

plugin ignores the changes, so choose a distinct account different from Directory Manager and other administrators.

1. Create or choose an account for the Samba Administrator:

```
$ cat samba.ldif
dn: uid=Samba Admin,ou=Special Users,dc=example,dc=com
cn: Samba Administrator
givenName: Samba
mail: samba@example.com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
sn: Administrator
uid: Samba Admin
userPassword: password

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

2. Ensure the Samba Administrator can reset user passwords:

```
$ cat samba-rights.ldif
dn: uid=Samba Admin,ou=Special Users,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: password-reset

dn: dc=example,dc=com
changetype: modify
add: aci
aci: (target="ldap:///dc=example,dc=com")(targetattr ="*")
 (version 3.0; acl "Samba Admin user rights"; allow(all)
  userdn="ldap:///uid=Samba Admin,ou=Special Users,dc=example,dc=com";)

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

To Set Up the Samba Password Plugin

1. Determine whether the plugin must store passwords hashed like LanManager (`sync-lm-password`) or like Windows NT (`sync-nt-password`), based on how you set up Samba in your environment.

2. Enable the plugin:

```
$ dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Samba Password Synchronisation" \
 --type samba-password \
 --set enabled:true \
 --set pwd-sync-policy:sync-nt-password \
 --set samba-administrator-dn:"uid=Samba Admin,ou=Special Users,dc=example,dc=com" \
 --trustAll \
 --no-prompt
```

At this point the Samba Password plugin is active.

3. (Optional) When troubleshooting Samba Password plugin issues, you can turn on debug logging as follows:

```
$ dsconfig \
 create-debug-target \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --target-name org.opends.server.plugins.SambaPasswordPlugin \
 --set enabled:true \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
$ tail -f /path/to/opendj/logs/debug
```

Chapter 19

Monitoring, Logging, and Alerts

This chapter covers monitoring capabilities. In this chapter you will learn to:

- Access monitoring information remotely using HTTP, LDAP, SNMP, or JMX.
- Monitor server status, including task status.
- Configure logs and interpret the messages they contain.
- Configure email settings for administrative alert notifications.

This chapter covers the options for monitoring servers.

HTTP-Based Monitoring

DS servers publish monitoring information at the endpoints described in "To Set Up REST Access to Administrative Data":

`/alive`

Provides an endpoint to check whether the server is currently *alive*, meaning that its internal checks have not found any errors that would require administrative action.

`/healthy`

Provides an endpoint to check whether the server is currently *healthy*, meaning that it is alive and any replication delays are below a configurable threshold.

`/metrics/api`

Provides a REST API to the server monitoring information with a read-only JSON-based view of `cn=monitor` and the monitoring backend.

Each LDAP entry maps to a resource under `/metrics/api`.

`/metrics/prometheus`

Provides an API to the server monitoring information for use with Prometheus monitoring software.

Many different types of information are exposed. For details, see "*Monitoring Metrics*" in the *Reference*.

The following example command accesses the Prometheus endpoint using the default monitoring account with the password set to `password`:

```
$ curl --user monitor:password http://opendj.example.com:8080/metrics/prometheus
```

You can optionally add the monitoring account and set the password when setting up the server. The default monitoring account has DN `uid=monitor`. The default configuration maps the HTTP username, `monitor`, to this LDAP account. For an example, see "To Set Up a Directory Server" in the *Installation Guide*.

This section includes the following examples:

- "Monitoring Liveness over HTTP"
- "Monitoring Ability to Handle Requests Over HTTP"
- "Monitoring Health Status With Prometheus"
- "Monitoring Replication Delay Over HTTP"
- "Monitoring Disk Space"
- "Monitoring Certificate Expiration"

Monitoring Liveness over HTTP

The following example reads the `/alive` endpoint anonymously. If the DS server's internal tests do not find errors that require administrative action, then it returns HTTP 200 OK:

```
$ curl --head http://opendj.example.com:8080/alive
HTTP/1.1 200 OK
Content-Length: 0
Date: <date>
```

If the server finds that it is subject to errors requiring administrative action, it returns HTTP 503 Service Unavailable.

If there are errors, anonymous users receive only the 503 error status. Error strings for diagnosis are returned as an array of `"alive-errors"` in the response body, but the response body is only returned to a user with the `monitor-read` privilege.

When a server returns `"alive-errors"`, diagnose and fix the problem, and then either restart or replace the server.

Monitoring Ability to Handle Requests Over HTTP

The following example reads the `/healthy` endpoint anonymously. If the DS server is alive as described in "Monitoring Liveness over HTTP", and any replication delay is below the threshold configured as `max-replication-delay-health-check` (default: 5 seconds), then it returns HTTP 200 OK:

```
$ curl --head http://opendj.example.com:8080/healthy
HTTP/1.1 200 OK
Content-Length: 0
Date: <date>
```

If the server is subject to a replication delay above the threshold, then it returns HTTP 503 Service Unavailable.

If there are errors, anonymous users receive only the 503 error status. Error strings for diagnosis are returned as an array of `"ready-errors"` in the response body, but the response body is only returned to a user with the `monitor-read` privilege.

When a server returns `"ready-errors"`, route traffic to another server until the current server is ready again.

Monitoring Health Status With Prometheus

In addition to the examples above, you can monitor whether a server is alive and able to handle requests as Prometheus metrics:

```
$ curl --user monitor:password http://opendj.example.com:8080/metrics/prometheus 2>/dev/null | grep
 health_status
# HELP ds_health_status_alive Indicates whether the server is alive
# TYPE ds_health_status_alive gauge
ds_health_status_alive 1.0
# HELP ds_health_status_healthy Indicates whether the server is able to handle requests
# TYPE ds_health_status_healthy gauge
ds_health_status_healthy 1.0
```

Monitoring Replication Delay Over HTTP

The following example reads a metric to check the delay in replication:

```
$ curl --user monitor:password http://opendj.example.com:8080/metrics/prometheus 2>/dev/null | grep
 current_delay
# TYPE ds_replication_replica_remote_replicas_current_delay_seconds gauge
ds_replication_replica_remote_replicas_current_delay_seconds{<labels>} <delay>
```

The server only begins updating and publishing the delay metric after it has received replication updates from other servers. The metric is therefore only present and meaningful after the server has received replication updates. The delay reflects the time between the latest update that the replica has received and the latest update that the replica has replayed. This metric is accurate only when the replica receives updates quickly. In the event of a network partition, the delay cannot accurately reflect updates happening on the other side of the partition.

Monitoring Disk Space

The following example shows monitoring metrics you can use to check whether the server is running out of disk space:

```
$ curl --user monitor:password http://opendj.example.com:8080/metrics/prometheus 2>/dev/null | grep disk
# HELP ds_disk_free_space_bytes The amount of free disk space (in bytes)
# TYPE ds_disk_free_space_bytes gauge
ds_disk_free_space_bytes{disk="<partition>",} <bytes>
# HELP ds_disk_free_space_full_threshold_bytes The effective full disk space threshold (in bytes)
# TYPE ds_disk_free_space_full_threshold_bytes gauge
ds_disk_free_space_full_threshold_bytes{disk="<partition>",} <bytes>
# HELP ds_disk_free_space_low_threshold_bytes The effective low disk space threshold (in bytes)
# TYPE ds_disk_free_space_low_threshold_bytes gauge
ds_disk_free_space_low_threshold_bytes{disk="<partition>",} <bytes>
```

In your monitoring software, compare free space with the disk low and disk full thresholds. For database backends, these thresholds are set using the configuration properties: disk-low-threshold and disk-full-threshold.

When you read from `cn=monitor` instead as described in "LDAP-Based Monitoring", the relevant data are exposed on child entries of `cn=disk space monitor,cn=monitor`.

Monitoring Certificate Expiration

The following example shows how you can use monitoring metrics to check whether the server certificate is due to expire soon:

```
$ curl --user monitor:password http://opendj.example.com:8080/metrics/prometheus 2>/dev/null | grep cert
# HELP ds_certificates_certificate_expires_at The certificate expiration date and time
# TYPE ds_certificates_certificate_expires_at gauge
ds_certificates_certificate_expires_at{alias="server-cert",key_manager="Default Key Manager",}
 <ms_since_epoch>
```

In your monitoring software, compare the expiration date with the current date.

When you read from `cn=monitor` instead as described in "LDAP-Based Monitoring", the relevant data are exposed on child entries of `cn=certificates,cn=monitor`.

LDAP-Based Monitoring

DS servers publish whether the server is alive and able to handle requests in the root DSE. They publish monitoring information over LDAP under the entry `cn=monitor`. The LDAP Schema Reference describes attributes and object classes used.

Many different types of information are exposed. For details, see "*Monitoring Metrics*" in the *Reference*.

The following example assigns the required privilege to Kirsten Vaughan's entry to read monitoring data, and shows monitoring information for the backend holding Example.com data:

```
$ cat monitor-read.ldif
dn: uid=kvaughan,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: monitor-read

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 monitor-read.ldif
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN cn=monitor \
 "(cn=dsEvaluation backend)"
dn: cn=dsEvaluation backend,cn=Disk Space Monitor,cn=monitor
objectClass: top
objectClass: ds-monitor-entry
objectClass: extensibleObject
disk-dir: /path/to/opendj/db/dsEvaluation
disk-free: <bytes>
disk-state: normal
cn: dsEvaluation backend

dn: cn=dsEvaluation Backend,cn=monitor
objectClass: top
objectClass: ds-monitor-entry
objectClass: ds-backend-monitor-entry
ds-backend-id: dsEvaluation
ds-backend-base-dn: dc=example,dc=com
ds-backend-is-private: false
ds-backend-entry-count: <count>
ds-base-dn-entry-count: <count> dc=example,dc=com
ds-backend-writability-mode: enabled
cn: dsEvaluation Backend
```

Notice that users must have the `monitor-read` privilege to read monitoring data.

As an alternative, you can add a monitoring account when setting up the server. For an example, see "To Set Up a Directory Server" in the *Installation Guide*.

This section includes the following examples:

- "Monitoring Health Status Anonymously Over LDAP"
- "Monitoring Health Status Details Over LDAP"
- "Monitoring Replication Delay Over LDAP"

Monitoring Health Status Anonymously Over LDAP

Anonymous clients can monitor the health status of the DS server by reading the `alive` attribute of the root DSE. The following example demonstrates this:

```
$ ldapsearch \
 --port 1389 \
 --baseDN "" \
 --searchScope base \
 "(&)" \
 alive
dn:
alive: true
```

When `alive` is `true`, the server's internal tests have not found any errors requiring administrative action. When it is `false`, fix the errors and either restart or replace the server.

If the server returns `false` for this attribute, get error information as described in "Monitoring Health Status Details Over LDAP".

Monitoring Health Status Details Over LDAP

The default monitor user can check whether the server is alive and able to handle requests on `cn=health status,cn=monitor`:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN "cn=health status,cn=monitor" \
 --searchScope base \
 "(&)"
dn: cn=health status,cn=monitor
ds-mon-alive: true
ds-mon-healthy: true
objectClass: top
objectClass: ds-monitor
objectClass: ds-monitor-health-status
cn: health status
```

When the server is either not alive or not able to handle requests, this entry includes error diagnostics as strings on the `ds-mon-alive-errors` and `ds-mon-healthy-errors` attributes.

Monitoring Replication Delay Over LDAP

The following example uses the default monitor user account to check the delay in replication:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN cn=monitor \
 "(ds-mon-current-delay=*)" \
 ds-mon-current-delay
dn: ds-mon-domain-name=cn=admin data,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: <delay>

dn: ds-mon-domain-name=cn=schema,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: <delay>

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

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

The server only begins updating and publishing the delay metric after it has received replication updates from other servers. The metric is therefore only present and meaningful after the server has received replication updates. The delay reflects the time between the latest update that the replica has received and the latest update that the replica has replayed. This metric is accurate only when the replica receives updates quickly. In the event of a network partition, the delay cannot accurately reflect updates happening on the other side of the partition.

SNMP-Based Monitoring

DS servers support SNMP, including the Management Information Base described in *RFC 2605: Directory Server Monitoring MIB*.

SNMP is not enabled by default. SNMP-based monitoring depends on an OpenDMK library. The OpenDMK binary bundle containing this library ships with DS servers as `snmp/opendmk.jar`. Installation requires that you accept the OpenDMK Binary License, and so OpenDMK installation is a separate step that you must perform before you can use SNMP.

To run the OpenDMK installer and accept the license, use the self-extracting .jar:

```
$ java -jar /path/to/opendj/snmp/opendmk.jar
```

Install OpenDMK, and then copy the libraries to the `/path/to/opendj/extlib` directory. For example, if you install OpenDMK in the `/path/to` directory, copy the libraries from the `/path/to/OpenDMK-bin/lib` directory:

```
$ cp /path/to/OpenDMK-bin/lib/* /path/to/opendj/extlib/
```

After installing OpenDMK, set up an SNMP connection handler:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name SNMP \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

By default, the SNMP connection handler listens on port 161 and uses port 162 for traps. On UNIX and Linux systems, only root can normally open these ports. To install as a normal user, change the listen and trap ports:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name SNMP \
 --set listen-port:11161 \
 --set trap-port:11162 \
 --trustAll \
 --no-prompt
```

Restart the SNMP connection handler to take the changes into account:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name SNMP \
 --set enabled:false \
 --trustAll \
 --no-prompt

$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name SNMP \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

To check that connection handler works as expected, use a command such as **snmpwalk** to read the response on the SNMP listen port:

```
$ snmpwalk -v 2c -c OpenDJ@OpenDJ localhost:11161
iso.3.6.1.2.1.66.1.1.1.1 = STRING: "ForgeRock Directory Services version"
iso.3.6.1.2.1.66.1.1.2.1 = STRING: "/path/to/opendj"
...
```

JMX-Based Monitoring

DS servers support JMX-based monitoring. A number of tools support JMX, including the **jconsole** and **jvisualvm** commands bundled with the Sun/Oracle Java platform. JMX is not configured by default.

Before enabling JMX, set server Java arguments appropriately to avoid regular full garbage collection events as described in "To Set Up JMX Access".

Use the **dsconfig** command to configure the JMX connection handler:

```
$ dsconfig \
 create-connection-handler \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name JMX \
 --type jmx \
 --set enabled:true \
 --set listen-port:1689 \
 --trustAll \
 --no-prompt
```

By default, no users have privileges to access the JMX connection. The following commands add JMX privileges for Directory Manager:

```
$ cat jmx-privileges.ldif
dn: cn=Directory Manager
changetype: modify
add: ds-privilege-name
ds-privilege-name: jmx-notify
ds-privilege-name: jmx-read
ds-privilege-name: jmx-write

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

Also configure security to login remotely. See the section on *Using SSL* in *Monitoring and Management Using JMX Technology* for hints.

Alternatively, connect to a local server process using the process ID:

```
$ jvisualvm --openpid $(</path/to/opendj/logs/server.pid)
```

Server Operation and Tasks

DS servers have commands for monitoring server processes and tasks. The **status** command, described in "*status — display basic OpenDJ server information*" in the *Reference*, displays basic information about the local server. The **manage-tasks** command, described in "*manage-tasks — manage server administration tasks*" in the *Reference*, lets you manage tasks scheduled on a server, such as nightly backup.

The **status** command takes administrative credentials to read the configuration:

```
$ status \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --hostname opendj.example.com \
 --port 4444 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --script-friendly
{
  "isRunning" : true,
  "hostname" : "opendj.example.com",
  "adminConnectorPort" : 4444,
  "openConnections" : <count>,
  "version" : "ForgeRock Directory Services 6.5.6",
  "installPath" : "/path/to/opendj",
  "instancePath" : "/path/to/opendj",
  "javaVersion" : "<version>",
  "javaVendor" : "<vendor>",
  "jvmCpus" : <count>,
  "jvmMaxHeapSizeBytes" : <size>,
  "connectionHandlers" : [ {
    "name" : "LDAP",
    "port" : 1389,
    "type" : "LDAP",
    "security" : "NONE",
    "enabled" : true,
    "avgLoadM1Rate" : <rate>,
    "avgLoadM5Rate" : <rate>
  }, {
    "name" : "LDAPS",
    "port" : 1636,
    "type" : "LDAPS",
    "security" : "SSL",
    "enabled" : true,
    "avgLoadM1Rate" : <rate>,
    "avgLoadM5Rate" : <rate>
  }, {
    "name" : "LDIF",
```

```
    "port" : null,
    "type" : "LDIF",
    "security" : null,
    "enabled" : false,
    "avgLoadM1Rate" : null,
    "avgLoadM5Rate" : null
  }, {
    "name" : "SNMP",
    "port" : 161,
    "type" : "SNMP",
    "security" : null,
    "enabled" : false,
    "avgLoadM1Rate" : null,
    "avgLoadM5Rate" : null
  } ],
  "backends" : [ {
    "name" : "rootUser",
    "type" : "LDIF",
    "enabled" : true,
    "dbCacheSizeBytes" : null,
    "baseDNs" : [ {
      "dn" : "cn=Directory Manager",
      "entryCount" : 1,
      "replicationStatus" : null,
      "replicationDelayMs" : null
    } ]
  }, {
    "name" : "dsEvaluation",
    "type" : "DB",
    "enabled" : true,
    "dbCacheSizeBytes" : <size>,
    "baseDNs" : [ {
      "dn" : "dc=example,dc=com",
      "entryCount" : <count>,
      "replicationStatus" : null,
      "replicationDelayMs" : null
    } ]
  } ],
  "disks" : [ {
    "root" : "<directory>",
    "state" : "normal",
    "freeSpaceRemainingBytes" : <size>
  } ]
}
```

The **manage-tasks** command connects to the administration port, and so can connect to both local and remote servers:

```
$ manage-tasks \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --trustAll \
 --no-prompt
```

Monitoring With Graphite

The Graphite application stores numeric time-series data of the sort produced by monitoring metrics, and allows you to render graphs of that data.

Your applications, in this case DS servers, push data into Graphite. You do this by configuring the "Graphite Monitor Reporter Plugin" with the host and port number of the Graphite service, and with a prefix for your server, such as its FQDN. By default, the plugin pushes all metrics it produces to the Graphite service. You can opt to limit this by setting the `excluded-metric-pattern` or `included-metric-pattern` properties.

The following example configures the plugin to push metrics to Graphite at `graphite.example.com:2004` every 10 seconds (default):

```
$ dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name Graphite \
 --type graphite-monitor-reporter \
 --set enabled:true \
 --set graphite-server:graphite.example.com:2004 \
 --set metric-name-prefix:opendj.example.com \
 --trustAll \
 --no-prompt
```

To view metrics stored in Graphite, you can use the Graphite render API or Grafana, for example. See the Graphite and Grafana documentation for details.

Server Logs

By default, the server stores the following files under the `logs/` directory:

- Access logs for messages about clients accessing the server.

  One form of access log is a directory audit log. An audit log records changes to directory data in LDIF.

- Debug logs for messages tracing internal server events.
- Error logs for messages tracing server events.
- Replication logs for messages used to help repair problems in data replication.
- A `server.out` log for messages about server events since startup.

Messages in this file have the same format as error log messages.

- A `server.pid` process ID file when the server is running.

Logs are handled by *log publishers*. Log publishers determine which messages to publish, where to publish them, and what output format to use.

The server's logging system supports extensibility through the ForgeRock Common Audit event framework. (Common Audit deals with any event you can audit, not only the directory data changes recorded in a directory audit log.) The ForgeRock Common Audit event framework provides log handlers for publishing to local files or to remote systems, as described in "Common ForgeRock Access Logs".

You configure logging using the **dsconfig** command. In addition to configuring how messages are published, you can configure policies for log file rotation and retention.

Access Logs

An *access log* traces client requests that the server processes. Each message includes a datestamp, information about the connection, and information about the operation processed.

Tip

It is possible to configure multiple access logs at the same time. This makes it possible to have a primary unfiltered access logger to record information about all client requests, and also additional filtered access logs as described below in "Access Log Filtering".

Do not enable multiple *unfiltered* access loggers, however.

An unfiltered access logger can put significant write load on the disk subsystem where access logs are stored. Every client request results in at least one new log message.

By default, the JSON file-based access logger is enabled. For details about this log publisher, see "Configuring JSON Access Logs".

Common ForgeRock Access Logs

DS servers support the ForgeRock Common Audit event framework, and uses the JSON file handler as the default logger. The log message formats are compatible for all products using the framework. The framework uses transaction IDs to correlate requests as they traverse the platform. This makes it easier to monitor activity and to enrich reports.

The ForgeRock Common Audit event framework is built on audit event handlers. Audit event handlers can encapsulate their own configurations. Audit event handlers are the same in each product in the ForgeRock platform. You can plug in custom handlers that comply with the framework without having to upgrade the server.

Note

The ForgeRock Common Audit event framework includes handlers for logging audit event messages to local files and facilities, as well as to remote systems.

Although the ForgeRock Common Audit event framework supports multiple topics, DS software currently supports handling only access events. DS software divides access events into `ldap-access` events and `http-access` events.

Common Audit transaction IDs are not recorded by default. In order to record transaction IDs in the access logs, you must configure the DS server to trust them as described in the sections below.

Common Audit LDAP events have the following format:

```
{
    "eventName": "DJ-LDAP",
    "client": {
        "ip": string,                       // Client IP address
        "port": number                      // Client port number
    },
    "server": {
        "ip": string,                       // Server IP address
        "port": number                      // Server port number
    },
    "request": {                            // LDAP request
        "attrs": [ string ],                // Requested attributes
        "authType": string,                 // Bind type such as "SIMPLE"
        "connId": number,                   // Connection ID
        "controls": string,                 // Request controls
        "deleteOldRDN": boolean,            // For a modify DN request
        "dn": string,                       // Bind DN
        "filter": string,                   // Search filter
        "idToAbandon": number,              // ID to use to abandon operation
        "message": string,                  // Localized request message
        "msgId": number,                    // Message ID
        "name": string,                     // Operation name
        "newRDN": string,                   // For a modify DN request
        "newSup": string,                   // For a modify DN request
        "oid": string,                      // Operation name or OID
        "operation": string,                // Examples: "CONNECT", "BIND", "SEARCH"
        "opType": "sync",                   // Replication operation
        "protocol": "LDAP",
        "runAs": string,                    // Authorization ID
        "scope": string,                    // Search scope such as "sub"
        "version": string                   // Version "2", "3"
    },
    "response": {
        "additionalItems": string           // Additional information
        "controls": string,                 // Response controls
        "elapsedTime": number,              // Number of time units
        "elapsedTimeUnits": string,         // Time unit such as "MILLISECONDS"
        "failureReason": string,            // Human-readable information
        "maskedMessage": string,            // Real, masked result message
        "maskedResult": string,             // Real, masked result code
        "nentries": number,                 // Number of entries returned
        "reason": string,                   // Reason for disconnect
```

```
        "status": string,                       // "SUCCESSFUL", "FAILED"
        "statusCode": string                    // For example, "0" for success
    },
    "timestamp": string,                        // UTC date
    "transactionId": string,                    // Unique ID for the transaction
    "userId": string,                           // User who requested the operation
    "_id": string                               // Unique ID for the operation
}
```

Common Audit HTTP events have the following format:

```
{
    "eventName": "DJ-HTTP",
    "client": {
        "ip": string,                           // Client IP address
        "port": number                          // Client port number
    },
    "server": {
        "ip": string,                           // Server IP address
        "port": number                          // Server port number
    },
    "http": {                                   // HTTP request and response
      "request": {
        "secure": boolean,                      // HTTP: false; HTTPS: true
        "method": string,                       // Examples: "GET", "POST", "PUT"
        "path": string,                         // URL
        "queryParameters": map,                 // map: { key-string: [ value-string ] }
        "cookies": map                          // map: { key-string: [ value-string ] }
      },
      "response": {
        "headers": map                          // map: { key-string: [ value-string ] }
      }
    },
    "response": {
        "detail": string,                       // Human-readable information
        "elapsedTime": number,                  // Number of time units
        "elapsedTimeUnits": string,             // Time unit such as "MILLISECONDS"
        "status": string,                       // "SUCCESSFUL", "FAILED"
        "statusCode": string                    // For example, "0" for success
    },
    "timestamp": string,                        // UTC date
    "transactionId": string,                    // Unique ID for the transaction
    "trackingIds": [ string ],                  // Unique IDs from the transaction context
    "userId": string,                           // User who requested the operation
    "_id": string                               // Unique ID for the operation
}
```

Configuring JSON Access Logs

A JSON handler sends messages to a JSON format file.

This section includes the following procedures:

- "To Configure JSON LDAP Access Logs"

- "To Configure Filtered JSON LDAP Access Logs"
- "To Enable JSON HTTP Access Logs"

To Configure JSON LDAP Access Logs

The primary JSON-based LDAP access log file is `logs/ldap-access.audit.json`. Primary access log files include messages for each LDAP operation. They can grow quickly, but are particularly useful for analyzing overall client behavior.

Follow these steps to change the configuration:

1. Decide whether to trust transaction IDs sent by client applications, used to correlate requests as they traverse multiple servers.

   Client applications using the ForgeRock Common Audit event framework send transaction IDs with their requests. The transaction IDs are used to correlate audit events for monitoring and reporting that trace the request through multiple applications.

   Transaction IDs are sent over LDAP using an internal DS request control. They are sent over HTTP in an HTTP header.

   By default, DS servers are configured not to trust transaction IDs sent with client application requests.

   If you configure the server or gateway to trust transaction IDs in client application requests, then outgoing requests reuse the incoming transaction ID. For each outgoing request in the transaction, the request's transaction ID has the form *`original-transaction-id/sequence-number`*, where *sequence-number* reflects the position of the request in the series of requests for this transaction. For example, if the *original-transaction-id* is `abc123`, the first outgoing request has transaction ID `abc123/0`, the second `abc123/1`, the third `abc123/2`, and so on. This helps you to distinguish specific requests within a transaction when correlating audit events from multiple services.

   To trust transactions, set the advanced global server property, `trust-transaction-ids`, to `true`:

   ```
   $ dsconfig \
    set-global-configuration-prop \
    --advanced \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --set trust-transaction-ids:true \
    --trustAll \
    --no-prompt
   ```

2. Edit the default access log publisher as necessary.

   The following example applies the default settings:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based Access Logger" \
 --set enabled:true \
 --add "rotation-policy:24 Hours Time Limit Rotation Policy" \
 --add "rotation-policy:Size Limit Rotation Policy" \
 --set "retention-policy:File Count Retention Policy" \
 --trustAll \
 --no-prompt
```

To Configure Filtered JSON LDAP Access Logs

In addition to the primary JSON-based LDAP access log, DS servers write messages to a filtered access log file, `logs/filtered-ldap-access.audit.json`. This log grows more slowly than the primary access log. It includes only messages about the following:

- Administrative requests related to backing up and restoring data, scheduling tasks, and reading and writing configuration settings
- Authentication failures
- Requests from client applications that are misbehaving
- Requests that longer than one second for the server to process
- Search requests that return more than 1000 entries
- Unindexed searches

Follow these steps to change the configuration:

1. Edit the filtered access log publisher as necessary.

   The following example updates the configuration to include control OIDs in log records:

   ```
   $ dsconfig \
    set-log-publisher-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --publisher-name "Filtered Json File-Based Access Logger" \
    --set log-control-oids:true \
    --trustAll \
    --no-prompt
   ```

2. (Optional) Edit the filtering criteria as necessary.

The following commands list the relevant default filtering criteria settings for the filtered access log:

```
$ dsconfig \
 get-access-log-filtering-criteria-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Filtered Json File-Based Access Logger" \
 --criteria-name "Administrative Requests" \
 --trustAll \
 --no-prompt
Property                              : Value(s)
--------------------------------------:--------------------------------------
log-record-type                       : add, bind, compare, delete, extended,
                                      : modify, rename, search
request-target-dn-equal-to            : "**,cn=backups", "**,cn=config",
                                      : "**,cn=tasks", cn=backups, cn=config,
                                      : cn=tasks
$ dsconfig \
 get-access-log-filtering-criteria-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Filtered Json File-Based Access Logger" \
 --criteria-name "Auth Failures" \
 --trustAll \
 --no-prompt
Property                              : Value(s)
--------------------------------------:--------------------------------------
log-record-type                       : add, bind, compare, delete, extended,
                                      : modify, rename, search
response-result-code-equal-to         : 7, 8, 13, 48, 49, 50, 123
$ dsconfig \
 get-access-log-filtering-criteria-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Filtered Json File-Based Access Logger" \
 --criteria-name "Long Requests" \
 --trustAll \
 --no-prompt
Property                              : Value(s)
--------------------------------------:--------------------------------------
log-record-type                       : add, bind, compare, delete, extended,
                                      : modify, rename, search
response-etime-greater-than           : 1000
$ dsconfig \
 get-access-log-filtering-criteria-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Filtered Json File-Based Access Logger" \
```

```
 --criteria-name "Misbehaving Clients" \
 --trustAll \
 --no-prompt
Property                              : Value(s)
--------------------------------------:--------------------------------------
log-record-type                       : add, bind, compare, delete, extended,
                                      : modify, rename, search
response-result-code-equal-to         : 1, 2, 14, 17, 18, 19, 21, 34, 60, 61,
                                      : 64, 65, 66, 67, 69
$ dsconfig \
 get-access-log-filtering-criteria-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Filtered Json File-Based Access Logger" \
 --criteria-name "Searches Returning 1000+ Entries" \
 --trustAll \
 --no-prompt
Property                              : Value(s)
--------------------------------------:---------
log-record-type                       : search
search-response-nentries-greater-than  : 1000
$ dsconfig \
 get-access-log-filtering-criteria-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Filtered Json File-Based Access Logger" \
 --criteria-name "Unindexed Searches" \
 --trustAll \
 --no-prompt
Property                              : Value(s)
--------------------------------------:---------
log-record-type                       : search
search-response-is-indexed            : false
```

For details about the LDAP result codes listed in the criteria, see "*LDAP Result Codes*" in the *Reference*.

For details about how filtering works, see "Access Log Filtering".

To Enable JSON HTTP Access Logs

When you enable the HTTP connection handler as described in "To Set Up REST Access to User Data", also consider enabling JSON-based HTTP access logs.

The default JSON-based HTTP access log file is `logs/http-access.audit.json`:

1. Decide whether to trust transaction IDs sent by client applications, used to correlate requests as they traverse multiple servers.

Client applications using the ForgeRock Common Audit event framework send transaction IDs with their requests. The transaction IDs are used to correlate audit events for monitoring and reporting that trace the request through multiple applications.

Transaction IDs are sent over LDAP using an internal DS request control. They are sent over HTTP in an HTTP header.

By default, DS servers are configured not to trust transaction IDs sent with client application requests.

If you configure the server or gateway to trust transaction IDs in client application requests, then outgoing requests reuse the incoming transaction ID. For each outgoing request in the transaction, the request's transaction ID has the form `original-transaction-id/sequence-number`, where *sequence-number* reflects the position of the request in the series of requests for this transaction. For example, if the *original-transaction-id* is `abc123`, the first outgoing request has transaction ID `abc123/0`, the second `abc123/1`, the third `abc123/2`, and so on. This helps you to distinguish specific requests within a transaction when correlating audit events from multiple services.

To trust transactions, set the advanced global server property, `trust-transaction-ids`, to `true`:

```
$ dsconfig \
 set-global-configuration-prop \
 --advanced \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set trust-transaction-ids:true \
 --trustAll \
 --no-prompt
```

2. Enable the log publisher:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based HTTP Access Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

Configuring CSV Access Logs

A CSV handler sends messages to a comma-separated variable (CSV) file.

Important

The CSV handler does not sanitize messages when writing to CSV log files.

Do not open CSV logs in spreadsheets and other applications that treat data as code.

This section includes the following procedures:

- "To Enable CSV LDAP Access Logs"
- "To Enable CSV HTTP Access Logs"

To Enable CSV LDAP Access Logs

The default CSV LDAP access log file is `logs/ldap-access.csv`:

1. Decide whether to trust transaction IDs sent by client applications, used to correlate requests as they traverse multiple servers.

   Client applications using the ForgeRock Common Audit event framework send transaction IDs with their requests. The transaction IDs are used to correlate audit events for monitoring and reporting that trace the request through multiple applications.

   Transaction IDs are sent over LDAP using an internal DS request control. They are sent over HTTP in an HTTP header.

   By default, DS servers are configured not to trust transaction IDs sent with client application requests.

   If you configure the server or gateway to trust transaction IDs in client application requests, then outgoing requests reuse the incoming transaction ID. For each outgoing request in the transaction, the request's transaction ID has the form *`original-transaction-id/sequence-number`*, where *sequence-number* reflects the position of the request in the series of requests for this transaction. For example, if the *original-transaction-id* is `abc123`, the first outgoing request has transaction ID `abc123/0`, the second `abc123/1`, the third `abc123/2`, and so on. This helps you to distinguish specific requests within a transaction when correlating audit events from multiple services.

   To trust transactions, set the advanced global server property, `trust-transaction-ids`, to `true`:

```
$ dsconfig \
 set-global-configuration-prop \
 --advanced \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set trust-transaction-ids:true \
 --trustAll \
 --no-prompt
```

2. Create an enabled CSV file access logger with optional rotation and retention policies as in the following example:

```
$ dsconfig \
 create-log-publisher \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Common Audit Csv File Access Logger" \
 --type csv-file-access \
 --set enabled:true \
 --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
 --set "rotation-policy:Size Limit Rotation Policy" \
 --set "retention-policy:File Count Retention Policy" \
 --trustAll \
 --no-prompt
```

3. (Optional) If you require tamper-evident logs, prepare a keystore as described in "To Prepare a Keystore for Tamper-Evident Logs". Then enable tamper-evident capability as in the following example:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Common Audit Csv File Access Logger" \
 --set tamper-evident:true \
 --set key-store-file:config/audit-keystore \
 --set key-store-pin:"&{file:config/audit-keystore.pin}" \
 --trustAll \
 --no-prompt
```

Tamper-evident logging relies on digital signatures and regularly flushing messages to the log system. In high-volume directory deployments with heavy access patterns, signing log messages has a severe negative impact on server performance, reducing throughput by orders of magnitude.

Be certain to test the performance impact of tamper-evident logging with realistic access patterns for your deployment before enabling the feature in production.

To Enable CSV HTTP Access Logs

If you have enabled the HTTP connection handler as described in "To Set Up REST Access to User Data", you might want to enable CSV-format HTTP access logs.

The default CSV HTTP access log file is `logs/http-access.csv`:

1. Decide whether to trust transaction IDs sent by client applications, used to correlate requests as they traverse multiple servers.

   Client applications using the ForgeRock Common Audit event framework send transaction IDs with their requests. The transaction IDs are used to correlate audit events for monitoring and reporting that trace the request through multiple applications.

   Transaction IDs are sent over LDAP using an internal DS request control. They are sent over HTTP in an HTTP header.

   By default, DS servers are configured not to trust transaction IDs sent with client application requests.

   If you configure the server or gateway to trust transaction IDs in client application requests, then outgoing requests reuse the incoming transaction ID. For each outgoing request in the transaction, the request's transaction ID has the form *`original-transaction-id/sequence-number`*, where *sequence-number* reflects the position of the request in the series of requests for this transaction. For example, if the *original-transaction-id* is `abc123`, the first outgoing request has transaction ID `abc123/0`, the second `abc123/1`, the third `abc123/2`, and so on. This helps you to distinguish specific requests within a transaction when correlating audit events from multiple services.

   To trust transactions, set the advanced global server property, `trust-transaction-ids`, to `true`:

   ```
   $ dsconfig \
    set-global-configuration-prop \
    --advanced \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --set trust-transaction-ids:true \
    --trustAll \
    --no-prompt
   ```

2. Create an enabled CSV file HTTP access logger with optional rotation and retention policies as in the following example:

```
$ dsconfig \
 create-log-publisher \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Common Audit Csv File HTTP Access Logger" \
 --type csv-file-http-access \
 --set enabled:true \
 --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
 --set "rotation-policy:Size Limit Rotation Policy" \
 --set "retention-policy:File Count Retention Policy" \
 --trustAll \
 --no-prompt
```

3. (Optional) If you require tamper-evident logs, prepare a keystore as described in "To Prepare a Keystore for Tamper-Evident Logs". Then enable tamper-evident capability as in the following example:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Common Audit Csv File HTTP Access Logger" \
 --set tamper-evident:true \
 --set key-store-file:config/audit-keystore \
 --set key-store-pin:"&{file:config/audit-keystore.pin}" \
 --trustAll \
 --no-prompt
```

Tamper-evident logging relies on digital signatures and regularly flushing messages to the log system. In high-volume directory deployments with heavy access patterns, signing log messages has a severe negative impact on server performance, reducing throughput by orders of magnitude.

Be certain to test the performance impact of tamper-evident logging with realistic access patterns for your deployment before enabling the feature in production.

Configuring Elasticsearch Access Logs

An Elasticsearch audit event handler sends messages to an Elasticsearch server.

Before you enable the Elasticsearch handler, create a mapping in the Elasticsearch server index for the messages. For a sample index definition, see `/path/to/opendj/config/audit-handlers/elasticsearch-index-setup-example.json`.

To enable the Elasticsearch handler, see "To Enable an Access Logger With an External Configuration". The JSON configuration file for the Elasticsearch handler has the following format:

```
{
  "class": "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
  "config": {
    "name": string,               // Handler name, such as "elasticsearch".
    "topics": [ string, ...],     // LDAP: "ldap-access"; HTTP: "http-access".
    "enabled": boolean,           // Is the handler enabled?
    "connection": {               // (Optional) Connect using default settings.
      "host": string,             // Elasticsearch host. Default: localhost.
      "port": number,             // Elasticsearch host. Default: 9200.
      "useSSL": boolean,          // Connect to Elasticsearch over HTTPS? Default: false.
      "username": string,         // (Optional) User name for HTTP Basic auth.
      "password": string          // (Optional) Password for HTTP Basic auth.
    },
    "indexMapping": {             // (Optional) No mapping specified.
      "indexName": string         // Name of the Elasticsearch index.
    },
    "buffering": {                // (Optional) Default: write each message separately, no buffering.
      "enabled": boolean,         // Buffer messages to be sent? Default: false.
      "maxSize": number,          // Maximum number of buffered events.
      "writeInterval": duration,  // Interval between sending batch of events.
      "maxBatchedEvents": number  // Number of events to send per interval.
    }
  }
}
```

For a sample configuration, see `/path/to/opendj/config/audit-handlers/elasticsearch-config.json-example`.

The `writeInterval` takes a duration.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days
- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds

- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Example: Elasticsearch For Access Log

This example demonstrates logging HTTP access messages to a local Elasticsearch server.

To prepare the example, complete these steps:

1. Install and run an Elasticsearch server on `localhost:9200`.
2. Create an `audit` index in the Elasticsearch server for HTTP audit event messages.

   The following command uses the example index configuration file:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data @/path/to/opendj/config/audit-handlers/elasticsearch-index-setup-example.json \
 http://localhost:9200/audit
{"acknowledged":true}
```

3. Configure DS servers to enable HTTP access as described in "To Set Up REST Access to User Data".
4. Add a JSON configuration file under for the handler:

```
$ cat /path/to/opendj/config/audit-handlers/elasticsearch-handler.json
{
  "class": "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
  "config": {
    "name": "elasticsearch",
    "topics": ["http-access"],
    "connection": {
      "useSSL": false,
      "host": "localhost",
      "port": 9200
    },
    "indexMapping": {
      "indexName": "audit"
    },
    "buffering": {
      "enabled": true,
      "maxSize": 10000,
      "writeInterval": "100 ms",
      "maxBatchedEvents": 500
    }
  }
}
```

5. Configure the server to use the Elasticsearch audit handler:

```
$ dsconfig \
create-log-publisher \
--hostname opendj.example.com \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--publisher-name "Elasticsearch HTTP Access Log Publisher" \
--type external-http-access \
--set enabled:true \
--set config-file:config/audit-handlers/elasticsearch-handler.json \
--trustAll \
--no-prompt
```

With Elasticsearch and the DS server running, audit event messages for HTTP requests to the DS server are sent to Elasticsearch.

The following example requests Babs Jensen's entry:

```
$ curl --user bjensen:hifalutin http://opendj.example.com:8080/api/users/bjensen
```

A search request to Elasticsearch shows the resulting audit event content:

```
$ curl 'localhost:9200/audit/_search?q=*&pretty'
```

See the Elasticsearch documentation for details on searching and search results.

Configuring JDBC Access Logs

A JDBC handler sends messages to an appropriately configured relational database table.

Before you enable the JDBC handler, create the necessary schema and tables in the target database. See the following example files:

- `/path/to/opendj/config/audit-handlers/mysql_tables-example.sql`
- `/path/to/opendj/config/audit-handlers/oracle_tables-example.sql`
- `/path/to/opendj/config/audit-handlers/postgres_tables-example.sql`

In addition, the JDBC handler depends on the JDBC driver for the database, and HirakiCP. Copy the JDBC driver .jar file for your database, the HirakiCP .jar file for your Java version, and any other dependent libraries required to the `/path/to/opendj/extlib/` directory.

To enable the JDBC handler, see "To Enable an Access Logger With an External Configuration". The JSON configuration file for the JDBC handler has the following format:

```
{
```

```
    "class": "org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler",
    "config": {
        "name": string,                   // Handler name, such as "jdbc".
        "topics": array,                  // LDAP: "ldap-access"; HTTP: "http-access".
        "databaseType": string,           // Supported by default: "h2", "mysql", "oracle", "postgres".
        "enabled": boolean,               // Is the handler enabled?
        "buffering": {                    // (Optional) Default: write each message separately, no
 buffering.
            "enabled": boolean,           // Buffer messages to be sent? Default: false.
            "writeInterval": duration,    // Duration; must be > 0 if buffering is enabled.
            "autoFlush": boolean,         // Flush messages automatically? Default: true.
            "maxBatchedEvents": number,   // Maximum messages in prepared statement. Default: 100.
            "maxSize": number,            // Maximum number of buffered messages. Default: 5000.
            "writerThreads": number       // Threads to write buffered messages: Default: 1.
        },
        "connectionPool": {
            "dataSourceClassName": string, // Either set this to the class name of the data source...
            "jdbcUrl": string,            // ...or set this to the JDBC URL to connect to the database.
            "username": string,           // Username to connect to the database.
            "password": string,           // Password to connect to the database.
            "autoCommit": boolean,        // (Optional) Commit transactions automatically? Default: true.
            "connectionTimeout": number,  // (Optional) Milliseconds to wait before timing out. Default:
 30,000.
            "idleTimeout": number,        // (Optional) Milliseconds to wait before timing out. Default:
 600,000.
            "maxLifetime": number,        // (Optional) Milliseconds thread remains in pool. Default:
 1,800,000.
            "minIdle": number,            // (Optional) Minimum connections in pool. Default: 10.
            "maxPoolSize": number,        // (Optional) Maximum number of connections in pool. Default:
 10.
            "poolName": string,           // (Optional) Name of connection pool. Default: audit.
            "driverClassName": string     // (Optional) Class name of database driver. Default: null.
        },
        "tableMappings": [                // Correspondence of message fields to database columns.
            {
                "event": string,          // LDAP: "ldap-access"; HTTP: "http-access".
                "table": string,          // LDAP: "ldapaccess"; HTTP: "httpaccess".
                "fieldToColumn": {        // Map of field names to database column names.
                    "event-field": "database-column" // Event-field takes JSON pointer.
                }
            }
        ]
    }
}
```

For a sample configuration, see `/path/to/opendj/config/audit-handlers/jdbc-config.json-example`.

The `writeInterval` takes a duration.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days
- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds
- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Configuring JMS Access Logs

A JMS handler is a JMS producer that publishes messages to an appropriately configured Java Message Service.

To enable the JMS handler, see "To Enable an Access Logger With an External Configuration". The JSON configuration file for the JMS handler has the following format:

```
{
    "class": "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
    "config": {
        "name": string,                    // Handler name, such as "jms".
        "enabled": boolean,                // Is the handler enabled?
        "topics": array,                   // LDAP: "ldap-access"; HTTP: "http-access".
        "deliveryMode": string,            // One of "NON_PERSISTENT", "PERSISTENT".
        "sessionMode": string,             // One of "AUTO", "CLIENT", "DUPS_OK".
        "batch": {                         // (Optional) Default: no batching.
            "batchEnabled": boolean,       // Batch messages to publish? Default: false.
            "capacity": number,            // Maximum capacity of publishing queue. Default: 1.
            "maxBatchedEvents": number,    // Maximum events to deliver in single publishing call.
 Default: 1.
            "threadCount": number,         // Worker threads to process publishing queue. Default: 1.
            "insertTimeoutSec": number,    // Seconds queue can block before adding new item. Default: 60.
            "pollTimeoutSec": number,      // Seconds worker threads wait for new item. Default: 10.
            "shutdownTimeoutSec": number   // Seconds publisher waits for threads at shutdown. Default:
 60.
        },
        "jndi": {                          // (Optional) Default: Use default settings.
            "connectionFactoryName": string, // JNDI name for JMS connection factory. Default:
 "ConnectionFactory".
            "topicName": string            // (Optional) Match the value in the context. Default: "audit".
            "contextProperties": {         // JNDI InitialContext properties.
                // These depend on the JNDI provider. See the provider documentation for details.
            }
        }
    }
}
```

For a sample configuration, see `/path/to/opendj/config/audit-handlers/jms-config.json-example`.

Sending JSON Access Logs to Standard Output

A JSON stdout handler sends messages to standard output.

Important

Only use this logger when running the server with **start-ds --noDetach**.

When running as a daemon without the `--noDetach` option, the server also logs the messages to the file, `/path/to/logs/server.out`. The server has no mechanism for rotating or removing the `server.out` log file, which is only cleared when the server starts.

As a result, using the JSON stdout handler when running the server without the `--noDetach` option can cause the server to eventually run out of disk space.

To enable the JSON stdout handler, see "To Enable an Access Logger With an External Configuration".

The JSON configuration file for the JSON stdout handler has the following format:

```
{
  "class": "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
  "config": {
    "enabled": boolean,                    // Is the handler enabled?
    "name": string,                        // Handler name, such as "json.stdout".
    "elasticsearchCompatible" : boolean,  // If true, the message ID field is named _eventId. (Default:
 _id)
    "topics": array,                       // LDAP: "ldap-access"; HTTP: "http-access".
  }
}
```

For a sample configuration, see `/path/to/opendj/config/audit-handlers/json-stdout-config.json-example`.

Configuring Splunk Access Logs

A Splunk handler sends messages to an appropriately configured Splunk service.

To enable the Splunk handler, see "To Enable an Access Logger With an External Configuration". The JSON configuration file for the Splunk handler has the following format:

```
{
    "class": "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
    "config": {
        "name": string,                    // Handler name, such as "splunk".
        "enabled": boolean,                // Is the handler enabled?
        "topics": array,                   // LDAP: "ldap-access"; HTTP: "http-access".
        "authzToken": string,              // Splunk authorization token for HTTP requests.
        "buffering": {                     // Required message buffering configuration.
            "maxBatchedEvents": number,    // Maximum messages in prepared statement.
            "maxSize": number,             // Maximum number of buffered messages.
            "writeInterval": duration      // Duration as described below.
        },
        "connection": {                    // (Optional) Default: Use default settings.
            "host": string,                // Splunk host name. Default: "localhost".
            "port": number,                // Splunk port number. Default: "8088".
            "useSSL": boolean              // Use secure connection to Splunk? Default: false.
        }
    }
}
```

For a sample configuration, see `/path/to/opendj/config/audit-handlers/splunk-config.json-example`.

The `writeInterval` takes a duration.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration

- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days
- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds
- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Configuring Access Logging to Syslog

A Syslog handler sends messages to the UNIX system log as governed by RFC 5424, *The Syslog Protocol*.

Note

The implementation currently only supports writing *access* messages to Syslog, rather than error messages. As a result, this feature is of limited use in most deployments.

To enable a Syslog handler, see "To Enable an Access Logger With an External Configuration". The JSON configuration file for the Syslog handler has the following format:

```
{
    "class": "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler",
    "config": {
        "name": string,            // Handler name, such as "syslog".
        "enabled": boolean,        // Default: false.
        "topics": array,           // LDAP: "ldap-access"; HTTP: "http-access".
        "protocol": string,        // "TCP" or "UDP".
        "host": string,            // Syslog daemon host, such as localhost; must resolve to IP address.
        "port": number,            // Syslog daemon port number, such as 514; range: 0 to 65535.
        "connectTimeout": number,  // If using TCP, milliseconds to wait before timing out.
        "facility": string,        // Syslog facility to use for event messages.
        "buffering": {             // (Optional) Default: write each message separately, no buffering.
            "enabled": boolean,    // Buffer messages to be sent? Default: false.
            "maxSize": number      // Maximum number of buffered messages. Default: 5000.
        }
    }
}
```

For a sample configuration, see `/path/to/opendj/config/audit-handlers/syslog-config.json-example`.

For additional details, see "Syslog Facility Values".

Syslog Facility Values

Value	Description
`kern`	Kernel messages.
`user`	User-level messages.
`mail`	Mail system.
`daemon`	System daemons.
`auth`	Security/authorization messages.
`syslog`	Messages generated internally by `syslogd`.
`lpr`	Line printer subsystem.
`news`	Network news subsystem.
`uucp`	UUCP subsystem.
`cron`	Clock daemon.
`authpriv`	Security/authorization messages.
`ftp`	FTP daemon.
`ntp`	NTP subsystem.
`logaudit`	Log audit.
`logalert`	Log alert.
`clockd`	Clock daemon.
`local0`	Local use 0.
`local1`	Local use 1.
`local2`	Local use 2.
`local3`	Local use 3.
`local4`	Local use 4.
`local5`	Local use 5.
`local6`	Local use 6.
`local7`	Local use 7.

Configuring Tamper-Evidence and External Handlers

This section includes the following procedures:

- "To Prepare a Keystore for Tamper-Evident Logs"
- "To Enable an Access Logger With an External Configuration"

To Prepare a Keystore for Tamper-Evident Logs

Tamper-evident logging depends on a public key/private key pair and on a secret key that are stored together in a JCEKS keystore. Follow these steps to prepare the keystore:

1. Create a password for the keystore.

   The following example uses the default file name. If you use a different filename, then you must edit the `key-store-pin` property when configuring the log publisher:

   ```
   $ touch /path/to/opendj/config/audit-keystore.pin
   $ chmod 600 /path/to/opendj/config/audit-keystore.pin
   # Add password in cleartext on the only line in the file:
   $ vi /path/to/opendj/config/audit-keystore.pin
   ```

2. Generate a key pair in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `Signature` for the signing key, where the key is generated with the `RSA` key algorithm and the `SHA256withRSA` signature algorithm.

   The following example uses the default file name. If you use a different filename, then you must edit `key-store-file` property when configuring the log publisher:

   ```
   $ keytool \
    -genkeypair \
    -keyalg RSA \
    -sigalg SHA256withRSA \
    -alias "Signature" \
    -dname "CN=opendj.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/opendj/config/audit-keystore \
    -storetype JCEKS \
    -storepass:file /path/to/opendj/config/audit-keystore.pin \
    -keypass:file /path/to/opendj/config/audit-keystore.pin
   ```

3. Generate a secret key in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `Password` for the symmetric key, where the key is generated with the `HmacSHA256` key algorithm and 256-bit key size.

   The following example uses the default file name. If you use a different filename, then you must edit `key-store-file` property when configuring the log publisher:

```
$ keytool \
 -genseckey \
 -keyalg HmacSHA256 \
 -keysize 256 \
 -alias "Password" \
 -keystore /path/to/opendj/config/audit-keystore \
 -storetype JCEKS \
 -storepass:file /path/to/opendj/config/audit-keystore.pin \
 -keypass:file /path/to/opendj/config/audit-keystore.pin
```

4. Verify that the keystore contains signature and password keys:

```
$ keytool \
 -list \
 -keystore /path/to/opendj/config/audit-keystore \
 -storetype JCEKS \
 -storepass:file /path/to/opendj/config/audit-keystore.pin
...
signature, <date>, PrivateKeyEntry,
<fingerprint>
password, <date>, SecretKeyEntry,
<fingerprint>
```

To Enable an Access Logger With an External Configuration

An access logger configuration that relies on a JSON configuration lets you use any Common Audit event handler, including customer handlers. The content of the configuration file depends on the audit event handler.

Follow these steps:

1. Decide whether to trust transaction IDs sent by client applications, used to correlate requests as they traverse multiple servers.

   Client applications using the ForgeRock Common Audit event framework send transaction IDs with their requests. The transaction IDs are used to correlate audit events for monitoring and reporting that trace the request through multiple applications.

   Transaction IDs are sent over LDAP using an internal DS request control. They are sent over HTTP in an HTTP header.

   By default, DS servers are configured not to trust transaction IDs sent with client application requests.

   If you configure the server or gateway to trust transaction IDs in client application requests, then outgoing requests reuse the incoming transaction ID. For each outgoing request in the transaction, the request's transaction ID has the form *`original-transaction-id/sequence-number`*, where *sequence-number* reflects the position of the request in the series of requests for this

transaction. For example, if the *original-transaction-id* is `abc123`, the first outgoing request has transaction ID `abc123/0`, the second `abc123/1`, the third `abc123/2`, and so on. This helps you to distinguish specific requests within a transaction when correlating audit events from multiple services.

To trust transactions, set the advanced global server property, `trust-transaction-ids`, to `true`:

```
$ dsconfig \
 set-global-configuration-prop \
 --advanced \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set trust-transaction-ids:true \
 --trustAll \
 --no-prompt
```

2. Create the external JSON configuration file for the handler.

   Base your work on the appropriate template in the `config/audit-handlers` directory.

3. (Optional) If this is a custom access logger provided separately, copy the custom handler .jar file to `/path/to/opendj/lib/extensions`.

4. Create a log publisher configuration for the access log, where the `type` defines whether the log contains messages about LDAP or HTTP requests:

   - For LDAP access logging, create an external access log publisher:

     ```
     $ dsconfig \
      create-log-publisher \
      --hostname opendj.example.com \
      --port 4444 \
      --bindDN "cn=Directory Manager" \
      --bindPassword password \
      --publisher-name "My External LDAP Access Log Publisher" \
      --type external-access \
      --set enabled:true \
      --set config-file:config/audit-handlers/handler-conf.json \
      --trustAll \
      --no-prompt
     ```

   - For HTTP access logging, create an external HTTP access log publisher:

```
$ dsconfig \
 create-log-publisher \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "My External HTTP Access Log Publisher" \
 --type external-http-access \
 --set enabled:true \
 --set config-file:config/audit-handlers/handler-conf.json \
 --trustAll \
 --no-prompt
```

Blacklisting Log Message Fields

In some cases, you might not want all available fields to appear in log messages. You can blacklist individual fields in common audit access logs to prevent the fields from appearing in messages. The following example prevents all request headers and query parameters from appearing in JSON HTTP access logs:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based HTTP Access Logger" \
 --set log-field-blacklist:/http/response/headers \
 --set log-field-blacklist:/http/response/queryParameters \
 --trustAll \
 --no-prompt
```

As shown in the example, the blacklist values are JSON paths to the fields in log messages. For CSV logs, the blacklist values map to the column headers, where the terms are separated by dots (`.`) rather than by slashes (`/`).

Native LDAP Access Logs

For LDAP connection handlers, you can configure a native format access log.

Tip

This format was previously the default for DS servers. This log format can be useful, for example, if you already have software configured to consume the messages.

This access log uses the File Based Access Log Publisher The default log file is `logs/access`.

The following command enables this LDAP access logger:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Access Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

By default, this access log contains a message for each request, and a message for each response, as well as messages for connection and disconnection. You can configure the access log to write messages only on responses by setting the property `log-format:combined`. The setting is useful when filtering messages based on response criteria. It causes the server to log one message per operation, rather than one message for the request and another for the response.

Native LDAP Audit Logs

An audit log is a type of access log that records changes to directory data in LDIF format.

Tip

When using replicated directory servers, another way of accessing changes is to use the external change log. The external change log includes changes for all servers in a replication topology, and identifies the user requesting each change.

For details, see "Change Notification For Your Applications".

This audit log uses the File Based Audit Log Publisher. The default log file is `logs/audit`.

The following command enables the default file-based audit logger:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Audit Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

To see audit log output, make a change to directory data. The following example changes a description:

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

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "uid=bjensen,ou=People,dc=example,dc=com" \
 --bindPassword hifalutin \
 bjensen-new-description.ldif
```

The audit log records the changes as shown in the following excerpt:

```
$ # <datestamp>; conn=<number>; op=<number>
dn: cn=File-Based Audit Logger,cn=Loggers,cn=config
changetype: modify
replace: ds-cfg-enabled
ds-cfg-enabled: true
-

# <datestamp>; conn=<number>; op=<number>
dn: uid=bjensen,ou=people,dc=example,dc=com
changetype: modify
add: description
description: New description
-
```

As stated above, audit logs record changes in LDIF format. This means that when an LDAP entry is deleted, the audit log records only its DN.

Standard HTTP Access Logs

For HTTP requests, you can configure an access logger that uses the Extended Log File Format, which is a W3C working draft.

This access log uses the File Based HTTP Access Log Publisher. The default log file is `logs/http-access`.

The following command enables this HTTP access logger:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based HTTP Access Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

The following default fields are shown here in the order they occur in the log file:

`cs-host`

Client host name

`c-ip`

Client IP address

`cs-username`

Username used to authenticate

`x-datetime`

Completion timestamp for the HTTP request, which you can configure using the `log-record-time-format` property

`cs-method`

HTTP method requested by the client

`cs-uri`

URI requested by the client

`cs-uri-stem`

URL-encoded path requested by the client

`cs-uri-query`

URL-encoded query parameter string requested by the client

`cs-version`

HTTP version requested by the client

`sc-status`

HTTP status code for the operation

`cs(User-Agent)`

User-Agent identifier

`x-connection-id`

Connection ID used for DS internal operations

When using this field to match HTTP requests with internal operations in the LDAP access log, first set the access log advanced property, `suppress-internal-operations`, to `false`. By default, internal operations do not appear in the LDAP access log.

`x-etime`

Execution time in milliseconds needed by DS to service the HTTP request

`x-transaction-id`

ForgeRock Common Audit event framework transaction ID for the request

This defaults to `0` unless you configure the server to trust transaction IDs.

Missing values are replaced with `-`. Tabs separate the fields, and if a field contains a tab character, then the field is surrounded with double quotes. DS software doubles double quotes in the field to escape them.

The following example shows an excerpt of an HTTP access log with space reformatted:

```
- <client-ip> bjensen  <datestamp> GET  /users/bjensen HTTP/1.1 200 <user-agent> 3  40
- <client-ip> bjensen  <datestamp> GET  /users/scarter HTTP/1.1 200 <user-agent> 4   9
- <client-ip> -        <datestamp> GET  /users/missing HTTP/1.1 401 <user-agent> 5   0
- <client-ip> kvaughan <datestamp> POST /users         HTTP/1.1 200 <user-agent> 6 120
```

You can configure the `log-format` for the access log using the **dsconfig** command.

In addition to the default fields, the following standard fields are supported:

`c-port`

Client port number

`s-computername`

Server name where the access log was written

`s-ip`

Server IP address

`s-port`

Server port number

Access Log Filtering

With the default access log configuration (no filtering), for every client application request, the server writes at least one message to its access log. This volume of logging gives you the information to analyze overall access patterns, or to audit access when you do not know in advance what you are looking for.

When you do know what you are looking for, log filtering lets you throttle logging to focus on what you want to see. You specify the criteria for a filtering policy, and apply the policy to a log publisher.

Log filtering policies use the following criteria:

- Client IP address, bind DN, group membership
- Operation type (abandon, add, bind, compare, connect, delete, disconnect, extended operation, modify, rename, search, and unbind)
- Port number
- Protocol used
- Response time
- Result codes (only log error results, for example)
- Search response criteria (number of entries returned, unindexed search, and others)
- Target DN
- User DN and group membership

A log publisher's filtering policy determines whether to include or exclude log messages that match the criteria.

Example: Exclude Administration-Related Messages

A common development troubleshooting technique consists of sending client requests while tailing the access log:

```
$ tail -f /path/to/opendj/logs/ldap-access.audit.json
```

When the **dsconfig** command accesses the configuration, access log messages are written for administrative operations. These messages can prevent you from seeing the messages of interest from client applications.

This example demonstrates how to filter access log messages for administrative connections over LDAPS on port 4444.

Configure access log filtering criteria:

```
$ dsconfig \
 create-access-log-filtering-criteria \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based Access Logger" \
 --criteria-name "Exclude LDAPS on 4444" \
 --type generic \
 --set connection-port-equal-to:4444 \
 --set connection-protocol-equal-to:ldaps \
 --trustAll \
 --no-prompt
```

Activate filtering to exclude administrative messages:

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based Access Logger" \
 --set filtering-policy:exclusive \
 --trustAll \
 --no-prompt
```

The publisher filters messages about administrative requests to port 4444.

Example: Audit Configuration Changes

This example demonstrates how to set up an audit log file to track changes to the server configuration. The LDAP representation of the configuration is found under the base DN `cn=config`.

As described in "Server Logs", an audit log is a type of access log that records changes to directory data in LDIF. The change records have timestamped comments with connection and operation IDs, so that you can correlate the changes with messages in access logs.

Create an audit log publisher:

```
$ dsconfig \
 create-log-publisher \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Server Configuration Audit Log" \
 --type file-based-audit \
 --set enabled:true \
 --set filtering-policy:inclusive \
 --set log-file:logs/config-audit \
 --set rotation-policy:"24 Hours Time Limit Rotation Policy" \
 --set rotation-policy:"Size Limit Rotation Policy" \
 --set retention-policy:"File Count Retention Policy" \
 --trustAll \
 --no-prompt
```

Create log filtering criteria for the logger that matches operations targeting `cn=config`:

```
$ dsconfig \
 create-log-publisher \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Server Configuration Audit Log" \
 --type file-based-audit \
 --set enabled:true \
 --set filtering-policy:inclusive \
 --set log-file:logs/config-audit \
 --set rotation-policy:"24 Hours Time Limit Rotation Policy" \
 --set rotation-policy:"Size Limit Rotation Policy" \
 --set retention-policy:"File Count Retention Policy" \
 --trustAll \
 --no-prompt
```

The server now writes to the current audit log file, `/path/to/opendj/logs/config-audit`, whenever an administrator changes the server configuration. The following example output show the resulting LDIF that defines the log filtering criteria:

```
# <datestamp>; conn=<id>; op=<id>
dn: cn=Record changes to cn=config,cn=Filtering Criteria,cn=File-Based Server Configuration Audit
 Log,cn=Loggers,cn=config
changetype: add
objectClass: top
objectClass: ds-cfg-access-log-filtering-criteria
cn: Record changes to cn=config
ds-cfg-request-target-dn-equal-to: **,cn=config
ds-cfg-request-target-dn-equal-to: cn=config
createTimestamp: <timestamp>
creatorsName: cn=Directory Manager
entryUUID: <uuid>
```

Debug Logs

A *debug log* traces internal server information needed to troubleshoot a problem. Debug logs can grow large quickly, and therefore no debug logs are enabled by default.

For debug logging, you must set a *debug target* to control what gets logged. For details, see "Enabling Debug Logging".

Error Logs

The *errors log* traces server events, error conditions, and warnings, categorized and identified by severity.

Messages in the `logs/errors` file have the following format:

```
[datestamp] category=category severity=severity msgID=ID number msg=message string
```

For lists of severe and fatal error messages by category, see the Log Message Reference.

Replication Repair Logs

The *replication repair log* traces replication events, with entries having the same format as the errors log:

```
[datestamp] category=SYNC severity=severity msgID=ID number msg=message string
```

The replication log does not trace replication operations. Use the external change log instead to get notifications about changes to directory data over protocol, as described in "Change Notification For Your Applications".

Log Rotation and Retention

Each file-based log can be associated with a *log rotation policy*, and a *log retention policy*. The rotation policy specifies when to rotate a log file based on a time, log file age, or log file size. The retention policy specifies whether to retain logs based on the number of logs, their size, or how much free space should be left on the disk.

Rotated logs have a rotation timestamp appended to their name.

You list existing log rotation policies with the **dsconfig list-log-rotation-policies** command, and retention policies with the **dsconfig list-log-retention-policies**:

```
$ dsconfig \
 list-log-rotation-policies \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --trustAll \
 --no-prompt
Log Rotation Policy                 : Type       : file-size-limit : rotation-interval : time-of-day
------------------------------------:------------:-----------------:--------------------:------------
24 Hours Time Limit Rotation Policy : time-limit : -               : 1 d                : -
7 Days Time Limit Rotation Policy   : time-limit : -               : 1 w                : -
Fixed Time Rotation Policy          : fixed-time : -               : -                  : 2359
Size Limit Rotation Policy          : size-limit : 100 mb          : -                  : -
$ dsconfig \
 list-log-retention-policies \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --trustAll \
 --no-prompt
Log Retention Policy             : Type            : disk-space-used : free-disk-space : number-of-files
---------------------------------:-----------------:-----------------:-----------------:----------------
File Count Retention Policy      : file-count      : -               : -               : 10
Free Disk Space Retention Policy : free-disk-space : -               : 500 mb          : -
Size Limit Retention Policy      : size-limit      : 500 mb          : -               : -
```

View the policies that apply for a given log with the **dsconfig get-log-publisher-prop** command. The following example shows that the server keeps 10 access log files, rotating either each day or when the log size reaches 100 MB:

```
$ dsconfig \
 get-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based Access Logger" \
 --property retention-policy \
 --property rotation-policy \
 --trustAll \
 --no-prompt
Property         : Value(s)
-----------------:---------------------------------------------------------------
retention-policy : File Count Retention Policy
rotation-policy  : 24 Hours Time Limit Rotation Policy, Size Limit Rotation
                 : Policy
```

Use the **dsconfig** command to create, update, delete, and assign log rotation and retention policies. Set the policy that applies to a logger with the **dsconfig set-log-publisher-prop** command.

Note

When using access logs based on the ForgeRock Common Audit event framework, such as the CSV file based or JSON file based access log publishers, you can only configure one of each type of retention or rotation policy.

In other words, you can configure one file count, free disk space, and size limit log retention policy, but not more than one of each. Also, you can configure one fixed time, size limit, and time limit log rotation policy, but not more than one of each.

Alert Notifications

DS servers can send notifications of significant server events. Alert notifications are not enabled by default.

The following example enables JMX alert notifications:

```
$ dsconfig \
 set-alert-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name "JMX Alert Handler" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

DS servers can send mail over SMTP instead of JMX notifications. Before you set up the SMTP-based alert handler, specify an SMTP server:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set smtp-server:opendj.example.com \
 --trustAll \
 --no-prompt
$ dsconfig \
 create-alert-handler \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name "SMTP Alert Handler" \
 --type smtp \
 --set enabled:true \
 --set message-subject:"DS Alert, Type: %%alert-type%%, ID: %%alert-id%%" \
 --set message-body:"%%alert-message%%" \
 --set recipient-address:kvaughan@example.com \
 --set sender-address:opendj@example.com \
 --trustAll \
 --no-prompt
```

Alert Types

DS servers use the following alert types. For alert types that indicate server problems, check `logs/errors` for details:

`org.opends.server.AccessControlDisabled`

The access control handler has been disabled.

`org.opends.server.AccessControlEnabled`

The access control handler has been enabled.

`org.opends.server.authentiation.dseecompat.ACIParseFailed`

The dseecompat access control subsystem failed to correctly parse one or more ACI rules when the server first started.

`org.opends.server.BackendRunRecovery`

The pluggable backend has thrown a `RunRecoveryException`. The server needs to be restarted.

`org.opends.server.CannotCopySchemaFiles`

A problem has occurred while attempting to create copies of the existing schema configuration files before making a schema update, and the schema configuration has been left in a potentially inconsistent state.

`org.opends.server.CannotRenameCurrentTaskFile`

The server is unable to rename the current tasks backing file in the process of trying to write an updated version.

`org.opends.server.CannotRenameNewTaskFile`

The server is unable to rename the new tasks backing file into place.

`org.opends.server.CannotScheduleRecurringIteration`

The server is unable to schedule an iteration of a recurring task.

`org.opends.server.CannotWriteConfig`

The server is unable to write its updated configuration for some reason and therefore the server may not exhibit the new configuration if it is restarted.

`org.opends.server.CannotWriteNewSchemaFiles`

A problem has occurred while attempting to write new versions of the server schema configuration files, and the schema configuration has been left in a potentially inconsistent state.

`org.opends.server.CannotWriteTaskFile`

The server is unable to write an updated tasks backing file for some reason.

`org.opends.server.DirectoryServerShutdown`

The server has begun the process of shutting down.

`org.opends.server.DirectoryServerStarted`

The server has completed its startup process.

`org.opends.server.DiskFull`

Free disk space has reached the full threshold.

Default is 6% of the size of the file system.

`org.opends.server.DiskSpaceLow`

Free disk space has reached the low threshold.

Default is 10% of the size of the file system.

`org.opends.server.EnteringLockdownMode`

The server is entering lockdown mode, wherein only root users are allowed to perform operations and only over the loopback address.

`org.opends.server.LDAPHandlerDisabledByConsecutiveFailures`

Consecutive failures have occurred in the LDAP connection handler and have caused it to become disabled.

`org.opends.server.LDAPHandlerUncaughtError`

Uncaught errors in the LDAP connection handler have caused it to become disabled.

`org.opends.server.LDIFBackendCannotWriteUpdate`

An LDIF backend was unable to store an updated copy of the LDIF file after processing a write operation.

`org.opends.server.LDIFConnectionHandlerIOError`

The LDIF connection handler encountered an I/O error that prevented it from completing its processing.

`org.opends.server.LDIFConnectionHandlerParseError`

The LDIF connection handler encountered an unrecoverable error while attempting to parse an LDIF file.

`org.opends.server.LeavingLockdownMode`

The server is leaving lockdown mode.

`org.opends.server.ManualConfigEditHandled`

The server detects that its configuration has been manually edited with the server online and those changes were overwritten by another change made through the server. The manually edited configuration will be copied to another location.

`org.opends.server.ManualConfigEditLost`

The server detects that its configuration has been manually edited with the server online and those changes were overwritten by another change made through the server. The manually edited configuration could not be preserved due to an unexpected error.

`org.opends.server.replication.UnresolvedConflict`

Multimaster replication cannot resolve a conflict automatically.

`org.opends.server.UncaughtException`

A server thread has encountered an uncaught exception that caused that thread to terminate abnormally. The impact that this problem has on the server depends on which thread was impacted and the nature of the exception.

`org.opends.server.UniqueAttributeSynchronizationConflict`

A unique attribute conflict has been detected during synchronization processing.

`org.opends.server.UniqueAttributeSynchronizationError`

An error occurred while attempting to perform unique attribute conflict detection during synchronization processing.

Chapter 20

Tuning Servers For Performance

This chapter suggests ways to measure and improve directory service performance. In this chapter you will learn to:

- Define directory service performance goals operationally in accordance with the needs of client applications
- Identify constraints that might limit achievable performance goals
- Design and execute appropriate performance tests with the help of DS command-line tools
- Adjust DS and system settings to achieve performance goals

Server tuning refers to the art of adjusting server, JVM, and system configuration to meet the service-level performance requirements of directory clients. In the optimal case you achieve service-level performance requirements without much tuning at all, perhaps only setting JVM runtime options when installing DS servers.

If you are reading this chapter, however, you are probably not facing an optimal situation. Instead you are looking for trade-offs that maximize performance for clients given the constraints of your deployment.

Defining Performance Requirements and Constraints

Your key performance requirement is most likely to satisfy your users or customers with the resources available to you. Before you can solve potential performance problems, define what those users or customers expect, and determine what resources you will have to satisfy their expectations.

Service Level Objectives

A service level objective (SLO) is a target for a directory service level that you can measure quantitatively. If possible, base SLOs on what your key users expect from the service in terms of performance.

You can define SLOs for many aspects of the directory service. You should have SLOs for at least the following areas:

- Directory service *response times*

Directory service response times range from less than a millisecond on average across a low latency connection on the same network to however long it takes your network to deliver the response. More important than average or best response times is the response time distribution, because applications set timeouts based on worst case scenarios. For example, a response time performance requirement might be defined as, *Directory response times must average less than 10 milliseconds for all operations except searches returning more than 10 entries, with 99.9% of response times under 40 milliseconds.*

- Directory service *throughput*

  Directory service throughput can range up to many thousands of operations per second. In fact there is no upper limit for read operations such as searches, because only write operations must be replicated. To increase read throughput, simply add additional replicas. More important than average throughput is peak throughput. You might have peak write throughput in the middle of the night when batch jobs update entries in bulk, and peak binds for a special event or first thing Monday morning. For example, a throughput performance requirement might be expressed as, *The directory service must sustain a mix of 5,000 operations per second made up of 70% reads, 25% modifies, 3% adds, and 2% deletes.*

  Even better is to mimic the behavior of key operations for performance testing, so that you understand the patterns of operations in the throughput you need to provide.

- Directory service *availability*

  DS software is designed to let you build directory services that are basically available, including during maintenance and even upgrade of individual servers. In order to reach very high levels of availability, you must make sure not only that the software supports availability, but also that your operations execute in such a way as to preserve availability. Availability requirements can be as lax as best effort, or as stringent as 99.999% or more uptime.

  Replication is the DS feature that allows you to build a highly available directory service.

- Directory service administrative support

  Be sure to understand how you support your users when they run into trouble. While directory services can help you turn password management into a self-service visit to a web site, some users still need to know what they can expect if they need your help.

Creating an SLO, even if your first version consists of guesses, helps you reduce performance tuning from an open-ended project to a clear set of measurable goals for a manageable project with a definite outcome.

Available Resources

With your SLOs in hand, inventory the server, networks, storage, people, and other resources at your disposal. Now is the time to estimate whether it is possible to meet the requirements at all.

If, for example you are expected to serve more throughput than the network can transfer, maintain high-availability with only one physical machine, store 100 GB of backups on a 50 GB partition, or provide 24/7 support all alone, no amount of tweaking available resources is likely to fix the problem.

When checking that the resources you have at least theoretically suffice to meet your requirements, do not forget that high availability in particular requires at least two of everything to avoid single points of failure. Be sure to list the resources you expect to have, when and how long you expect to have them, and why you need them. Also make note of what is missing and why.

In addition to the suggestions in this section, also read "Choosing Hardware" in the *Release Notes*.

Server Hardware Recommendations

DS servers are pure Java applications, making them very portable. DS servers tend to perform best on single-board, x86 systems due to low memory latency.

Advice Concerning Storage

High-performance storage is essential for handling high-write throughput. When the database stays fully cached in memory, directory read operations do not result in disk I/O. Only writes result in disk I/O. You can further improve write performance by using solid-state disks for persistent storage, or for file system cache.

Warning

DS directory servers are designed to work with *local storage* for database backends. *Do not use network file systems, such as NFS, where there is no guarantee that a single process has access to files.*

Storage area networks (SANs) and attached storage are fine for use with DS directory servers.

Regarding database size on disk, sustained write traffic can cause the database to grow to more than twice its initial size on disk. This is normal behavior. The size on disk does not impact the DB cache size requirements.

In order to avoid directory database file corruption after crashes or power failures on Linux systems, enable file system write barriers and make sure that the file system journaling mode is ordered. For details on how to enable write barriers and how to set the journaling mode for data, see the options for your file system in the **mount** command manual page.

Testing Performance

Even if you do not need high availability, you still need two of everything, because your test environment needs to mimic your production environment as closely as possible if you want to avoid unwelcome surprises.

In your test environment, you set up DS servers as you will later in production, and then conduct experiments to determine how to best meet your SLOs.

Use the **makeldif** command, described in "*makeldif — generate test LDIF*" in the *Reference*, to generate sample data that match what you expect to find in production.

The following command-line tools help with basic performance testing:

- The **addrate** command, described in "*addrate — measure add and delete throughput and response time*" in the *Reference*, measures add and delete throughput and response time.
- The **authrate** command, described in "*authrate — measure bind throughput and response time*" in the *Reference*, measures bind throughput and response time.
- The **modrate** command, described in "*modrate — measure modification throughput and response time*" in the *Reference*, measures modification throughput and response time.
- The **searchrate** command, described in "*searchrate — measure search throughput and response time*" in the *Reference*, measures search throughput and response time.

All these commands show you information about the response time distributions, and allow you to perform tests at specific levels of throughput.

If you need additional precision when evaluating response times, use the global configuration setting etime-resolution, to change elapsed processing time resolution from milliseconds (default) to nanoseconds:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set etime-resolution:nanoseconds \
 --trustAll \
 --no-prompt
```

The `etime`, recorded in the server access log, indicates the elapsed time to process the request, starting from the moment the decoded operation is available to be processed by a worker thread.

Tweaking DS Performance

When your tests show that DS performance is lacking even though you have the right underlying network, hardware, storage, and system resources in place, you can tweak DS performance in a number of ways. This section covers the most common tweaks.

Setting Maximum Open Files

DS servers need to be able to open many file descriptors, especially when handling thousands of client connections. Linux systems in particular often set a limit of 1024 per user, which is too low to handle many client connections to the DS server.

When setting up your DS server for production use, make sure the server can use at least 64K (65536) file descriptors. For example, when running the server as user `opendj` on a Linux system that uses `/etc/security/limits.conf` to set user level limits, you can set soft and hard limits by adding these lines to the file:

```
opendj soft nofile 65536
opendj hard nofile 131072
```

The example above assumes the system has enough file descriptors available overall. You can check the Linux system overall maximum as follows:

```
$ cat /proc/sys/fs/file-max
204252
```

Linux Page Caching

Default Linux virtual memory settings cause significant buildup of dirty data pages before they are flushed. When the kernel finally flushes the pages to disk, the operation can exhaust the disk I/O for up to several seconds. Application operations waiting on the file system to synchronize to disk are blocked.

The default virtual memory settings can therefore cause DS server operations to block for seconds at a time. Symptoms include high outlier etimes, even for very low average etimes.

To avoid these problems, tune Linux page caching. As a starting point for testing and tuning, set `vm.dirty_background_bytes` to one quarter of the disk I/O per second, and `vm.dirty_expire_centisecs` to 1000 (10 seconds) using the **sysctl** command. This causes the kernel to flush more often, and limits the pauses to a maximum of 250 milliseconds.

For example, if the disk I/O is 80 MB/second for writes, the following example shows an appropriate starting point. It updates the `/etc/sysctl.conf` file to change the setting permanently, and uses the **sysctl -p** command to reload the settings:

```
$ echo vm.dirty_background_bytes=20971520 | sudo tee -a /etc/sysctl.conf
[sudo] password for admin:
$ echo vm.dirty_expire_centisecs=1000 | sudo tee -a /etc/sysctl.conf
$ sudo sysctl -p
vm.dirty_background_bytes = 20971520
vm.dirty_expire_centisecs = 1000
```

Be sure to test and adjust the settings for your deployment.

For additional details, see the Oracle documentation on *Linux Page Cache Tuning*, and the Linux **sysctl** command virtual memory kernel reference, https://www.kernel.org/doc/Documentation/sysctl/vm.txt.

Java Settings

Default Java settings let you evaluate DS servers using limited system resources. If you need high performance for production system, test with the following JVM options.

Tip

To apply JVM settings for your server, edit `config/java.properties`.

The following **java** options are available in at least Oracle Java and OpenJDK:

`-server`

Use the C2 compiler and optimizer (HotSpot Server VM) when running Java in 32-bit mode.

`-Xmn`

When using CMS garbage collection, consider using this option. Do not use it when using G1 garbage collection (default in Java 11).

If a server handles high throughput, set the new generation size large enough for the JVM to avoid promoting short-lived objects into the old generation space (`-Xmn512M`).

`-Xms, -Xmx`

Set both minimum and maximum heap size to the same value to avoid resizing. Leave space for the entire DB cache and more.

Use at least a 2 GB heap (`-Xms2G -Xmx2G`) unless your data set is small.

`-XX:+DisableExplicitGC`

When using JMX, add this option to the list of `start-ds.java-args` arguments to avoid periodic full GC events.

JMX is based on RMI, which uses references to objects. By default, the JMX client and server perform a full GC periodically to clean up stale references. As a result, the default settings cause JMX to cause a full GC every hour.

Avoid using this argument with `import-ldif.offline.java-args` or when using the **import-ldif** command. The import process uses garbage collection to manage memory and references to memory-mapped files.

`-XX:MaxTenuringThreshold=1`

Force the server to only create objects that have either a short lifetime, or a long lifetime.

`-XX:+PrintGCDetails` (Java 8 only)
`-XX:+PrintGCTimeStamps` (Java 8 only)
`-Xlog:gc* (Java 11 only)`

Use these options when diagnosing JVM tuning problems. You can turn them off when everything is running smoothly.

When using the `-Xlog:gc` option, also specify the level and output file as necessary. For example, **-Xlog:gc=info:file=gc.log** logs informational messages about garbage collection to a `gc.log` file. By default, messages are logged to standard output. See **java -Xlog:help** for details.

`-XX:TieredStopAtLevel=1` (Java 11 only)

Short-lived client tools, such as the **ldapsearch** command, start up faster when this option is set to `1` as shown.

`-XX:+UseCompressedOops`

Set this option when you have a 64-bit JVM, and `-Xmx` less than 32 GB. Java object pointers normally have the same size as native machine pointers. If you run a small 64-bit JVM, then compressed object pointers can save space.

`-XX:+UseConcMarkSweepGC`

In Java 8, the CMS garbage collector tends to give the best performance characteristics with the lowest garbage collection pause times. Consider using the G1 garbage collector only if CMS performance characteristics do not fit your deployment, and testing shows G1 performs better.

In Java 11, the G1 garbage collector is the default. The CMS garbage collector is deprecated.

Data Storage Settings

By default, DS servers compress attribute descriptions and object class sets to reduce data size. This is called compact encoding.

By default, DS servers do not, however, compress entries stored in its backend database. If your entries hold values that compress well—such as text— you can gain space by setting the backend property `entries-compressed` to `true` before you (re-)import data from LDIF. With `entries-compressed: true` The DS server compresses entries before writing them to the database:

```
$ dsconfig \
 set-backend-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set entries-compressed:true \
 --trustAll \
 --no-prompt
$ import-ldif \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --ldifFile backup.ldif \
 --backendID dsEvaluation \
 --includeBranch dc=example,dc=com \
 --trustAll
```

DS directory servers do not proactively rewrite all entries in the database after you change the settings. Instead, to force the DS server to compress all entries, import the data from LDIF.

LDIF Import Settings

You can tweak DS servers to speed up import of large LDIF files.

By default, the temporary directory used for scratch files is `import-tmp` under the directory where you installed the DS server. Use the **import-ldif** command, described in "*import-ldif — import directory data from LDIF*" in the *Reference*, with the `--tmpdirectory` option to set this directory to a `tmpfs` file system, such as `/tmp`.

If you are certain your LDIF contains only valid entries with correct syntax, because the LDIF was exported with all checks active, for example, you can skip schema validation. Use the `--skipSchemaValidation` option with the **import-ldif** command.

Database Cache Settings

By default, DS directory servers use shared cache for all JE database backends. For details, see `je-backend-shared-cache-enabled`. The recommended setting is to leave this global property set to `true`.

If your deployment requires fine-grained control over JE backend cache settings, you can configure the amount of memory requested for database cache per database backend:

1. Configure `db-cache-percent` or `db-cache-size` per JE backend.

   The settings are described below.

2. Set the global property `je-backend-shared-cache-enabled` to `false`.

3. Restart the server for the changes to take effect.

`db-cache-percent`

Percentage of JVM memory to allocate to the database cache for the backend.

If the directory server has multiple database backends, the total percent of JVM heap used must remain less than 100 (percent), and must leave space for other uses.

Default: 50 (percent)

`db-cache-size`

JVM memory to allocate to the database cache.

This is an alternative to `db-cache-percent`. If you set its value larger than 0, then it takes precedence over `db-cache-percent`.

Default: 0 MB

Use default settings if the server has only one user data backend and its JVM heap is 2 GB or less. Otherwise, allocate a larger percentage of heap space to DB cache.

Depending on the size of your database, you have a choice to make about database cache settings:

- By caching the entire database in the JVM heap, you can get more deterministic response times and limit disk I/O. Yet, caching the whole DB can require a very large JVM.
- You can use a smaller JVM heap by properly tuning the JVM, and by allowing file system cache to hold the portion of database that does not fit in the DB cache. How you configure the file system cache depends on your operating system.

For very large database backends holding hundreds of millions of entries, default settings can potentially limit performance. As described in "About Database Backends", a JE backend stores data on disk in append-only log files. The maximum size of each log file is configurable. A JE backend keeps a configurable maximum number of log files open, caching file handles to the log files.

The relevant JE backend settings are the following:

`db-log-file-max`

Maximum size of a database log file.

Default: 1 GB

`db-log-filecache-size`

File handle cache size for database log files.

Default: 200

Given the defaults, if the size of the database reaches 200 GB on disk (1 GB x 200 files), the JE backend must start closing one log file in order to open another. This has serious impact on performance when the file cache starts to thrash. Having the JE backend open and close log files from time to time is okay.

To avoid this situation, increase `db-log-filecache-size` until the JE backend can cache file handles to all its log files. When changing the settings, also make sure that the maximum number of open files is sufficient. For details, see "Setting Maximum Open Files".

With large databases, such as a backend holding 10 million entries or more, the system might not have enough space to cache all data in memory. Even if memory space is available, the resulting JVM can grow so large that operations such as garbage collection and heap dumps become very costly. If you cannot or decide not to cache all data in a backend, at least cache all *internal nodes*. The following paragraphs explain what this means, and how to achieve this goal.

As described in "About Database Backends", a JE backend is implemented as a B-tree data structure. A B-tree is made up of nodes that can have children. Nodes with children are called internal nodes. Nodes without children are called *leaf nodes*.

The directory stores data in key-value pairs. Internal nodes hold the keys. Leaf nodes hold the values. One internal node usually holds keys to values in many leaf nodes. A B-tree has many more leaf nodes than internal nodes.

In order to read a value by its key, the backend traverses all internal nodes on the branch from the B-tree root to the leaf node holding the value. The backend is more likely to access nodes the closer they are to the B-tree root. An internal node immediately below the root node is superior

to approximately 1% of leaf nodes. An internal node two levels below the root node is superior to ~1% of ~1% of leaf nodes (~0.01%). Internal nodes are therefore accessed far more frequently than leaf nodes, and so should remain cached in memory. In addition to the worker threads serving client application requests, cleaner threads working in the background also access internal nodes frequently. The performance impact of having to fetch frequently used internal nodes from disk can be severe.

After cache memory fills, the backend must begin evicting nodes from cache in order to load others. The backend evicts leaf nodes first, because it is less likely to access a leaf node than an internal node. If, however, the internal nodes in use do not all fit in cache, then the backend must eventually evict such critical internal nodes.

The rest of this section describes how to estimate minimum DB cache size, and how to monitor backends for evictions of critical internal nodes. The examples below reflect a directory server with a 10 million-entry `userRoot` backend originally set up to be empty except for the base DN, `dc=example,dc=com`. The backend holds generated Example.com entries from the default template edited so that `define numusers=10000000`. The backend has the default indexes listed in "Default Indexes".

Base your own calculations on realistic sample data, with the same indexes that you use in production, and with data affected by realistic client application and replication loads. To generate your own sample data, start by reading "Generating Test Data". To simulate load, use the tools described in "Testing Performance". Even better, learn about real loads from analysis of production access logs, and build custom test clients that reflect the access patterns of your applications.

Note

After using the **import-ldif** command, the backend contains the minimum number of internal nodes required for the data. Over time as external applications update the directory server, the number of internal nodes grows. A JE backend only appends to the database log for update operations, so many internal nodes in the database logs of a live system represent garbage that the backend eventually cleans up. Only the live internal nodes must be cached in memory. Over time, the increase in the number of internal nodes should track backend growth.

After loading the server for some time, stop the server, and use the **backendstat** command and JE `DbCacheSize` tool together to estimate the required DB cache size. The following example uses the **backendstat** command to discover information about keys in the backend. Using a script or a spreadsheet on the output, calculate the total number of keys (sum of Total Keys, here: 73253009) and average key size (sum of Key Size/sum of Total Keys, here: nearly 14). Then use the results as input to the JE `DbCacheSize` tool:

```
# Stop the server before using backendstat:
$ stop-ds
$ backendstat list-raw-dbs --backendId userRoot
Raw DB Name                                                    Total Keys  Keys Size  Values Size
 Total Size
-------------------------------------------------------------------------------------------------------------------------------
/compressed_schema/compressed_attributes                       26          26         318
 344
/compressed_schema/compressed_object_classes                   4           4          121
 125
/dc=com,dc=example/aci.presence                                0           0          0            0
/dc=com,dc=example/cn.caseIgnoreMatch                          10000000    139240357  49902855
 189143212
```

```
/dc=com,dc=example/cn.caseIgnoreSubstringsMatch:6                    858259      5103773     205357681
 210461454
/dc=com,dc=example/dn2id                                             10000002    268888900   80000016
 348888916
/dc=com,dc=example/ds-sync-conflict.distinguishedNameMatch           0           0           0           0
/dc=com,dc=example/ds-sync-hist.changeSequenceNumberOrderingMatch    0           0           0           0
/dc=com,dc=example/entryUUID.uuidMatch                               9988339     39953356    49886797
 89840153
/dc=com,dc=example/givenName.caseIgnoreMatch                         8605        51628       20025815
 20077443
/dc=com,dc=example/givenName.caseIgnoreSubstringsMatch:6             19629       97542       48330063
 48427605
/dc=com,dc=example/id2childrencount                                  35          225         47
 272
/dc=com,dc=example/id2entry                                          10000002    80000016    5424234563
 5504234579
/dc=com,dc=example/mail.caseIgnoreIA5Match                           10000000    238888890   49902855
 288791745
/dc=com,dc=example/mail.caseIgnoreIA5SubstringsMatch:6               1222232     7333377     113418251
 120751628
/dc=com,dc=example/member.distinguishedNameMatch                     0           0           0           0
/dc=com,dc=example/objectClass.objectIdentifierMatch                 6           66          12
 78
/dc=com,dc=example/referral                                          0           0           0           0
/dc=com,dc=example/sn.caseIgnoreMatch                                13419       92727       20040257
 20132984
/dc=com,dc=example/sn.caseIgnoreSubstringsMatch:6                    41523       219199      73753666
 73972865
/dc=com,dc=example/state                                             18          869         18
 887
/dc=com,dc=example/telephoneNumber.telephoneNumberMatch              9989800     109887800   49888732
 159776532
/dc=com,dc=example/telephoneNumber.telephoneNumberSubstringsMatch:6  1111110     6543210     222040573
 228583783
/dc=com,dc=example/uid.caseIgnoreMatch                               10000000    118888890   49902855
 168791745
/dc=com,dc=example/uniqueMember.uniqueMemberMatch                    0           0           0           0

 Total: 25

# Calculate sum of Total Keys, sum of Key Size, and average key size.
$ java -cp /path/to/opendj/lib/opendj.jar com.sleepycat.je.util.DbCacheSize -records 73253009 -key 14

=== Environment Cache Overhead ===

3,158,477 minimum bytes

To account for JE daemon operation, record locks, HA network connections, etc,
a larger amount is needed in practice.

=== Database Cache Size ===

Number of Bytes  Description
---------------  -----------
2,815,557,912  Internal nodes only

To get leaf node sizing specify -data
```

```
For further information see the DbCacheSize javadoc.
```

The resulting recommendation for DB cache size, 2,815,557,912 bytes in this case, is a minimum estimate. Round up when configuring backend settings for `db-cache-percent` or `db-cache-size`. If the system in this example has 8 GB available memory, then you could leave the default setting of `db-cache-percent: 50`. (50% * 8 GB = 4 GB, which is larger than the minimum estimate.)

After deploying servers with properly sized DB caches, monitor the backend database environment information to determine whether each backend has evicted internal nodes. The following example has no internal node (IN) evictions:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "uid=monitor" \
 --bindPassword password \
 --baseDN ds-cfg-backend-id=userRoot,cn=backends,cn=monitor \
 "(ds-mon-db-cache-evict-internal-nodes-count=*)" \
 ds-mon-db-cache-evict-internal-nodes-count

dn: ds-cfg-backend-id=userRoot,cn=backends,cn=monitor
ds-mon-db-cache-evict-internal-nodes-count: 0
```

If `ds-mon-db-cache-evict-internal-nodes-count` is not `0`, then the system has too little memory for all internal nodes to remain in DB cache. Increase the amount of memory available for DB cache, and repeat the tuning process described above.

Caching Large, Frequently Used Entries

DS servers implement an entry cache designed for deployments with a few large entries that are regularly updated or accessed. The common use case is a deployment with a few large static groups that are updated or accessed regularly. An entry cache is used to keep such groups in memory in a format that avoids the need to constantly read and deserialize the large entries.

When configuring an entry cache, take care to include only the entries that need to be cached by using the configuration properties `include-filter` and `exclude-filter`. The memory devoted to the entry cache is not available for other purposes.

The following example adds a Soft Reference entry cache to hold entries that match the filter `(ou=Large Static Groups)`. A Soft Reference entry cache allows cached entries to be released if the JVM is running low on memory. A Soft Reference entry cache has no maximum size setting, so the number of entries cached is limited only by the `include-filter` and `exclude-filter` settings:

```
$ dsconfig \
 create-entry-cache \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --cache-name "Large Group Entry Cache" \
 --type soft-reference \
 --set cache-level:1 \
 --set include-filter:"(ou=Large Static Groups)" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

The entry cache configuration takes effect when the entry cache is enabled.

Logging Settings

Debug logs trace the internal workings of DS servers, and generally should be used sparingly. Be particularly careful when activating debug logging in high-performance deployments.

In general, leave other logs active for production environments to help troubleshoot any issues that arise.

For DS servers handling very high throughput, however, such as 100,000 operations per second or more, the access log constitute a performance bottleneck. Each client request results in at least one access log message. Consider disabling the access log in such cases.

The following command disables the JSON-based LDAP access logger described in "Configuring JSON Access Logs":

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "Json File-Based Access Logger" \
 --set enabled:false \
 --trustAll \
 --no-prompt
```

The following command disables the access logger described in "Standard HTTP Access Logs":

```
$ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based HTTP Access Logger" \
 --set enabled:false \
 --trustAll \
 --no-prompt
```

Changelog Settings

By default, a replication server indexes change numbers for replicated user data. This allows legacy applications to get update notifications by change number, as described in "To Align Draft Change Numbers". Indexing change numbers requires additional CPU, disk accesses and storage, so it should not be used unless change number-based browsing is required.

Disable change number indexing if it is not needed. For details, see "To Disable Change Number Indexing".

Chapter 21

Securing and Hardening Servers

When you deploy DS servers in production, there are specific precautions you should take to minimize risks. This chapter recommends the key precautions to take. In this chapter you will learn to:

- Set up a special system account for the DS server, and appropriately protect access to server files
- Enforce use of the latest Java security updates
- Enable only directory services that are actually used
- Use appropriate log configuration, global access control, password storage, and password policy settings
- Avoid overuse of the default directory root user account
- Use appropriate global access control settings
- Secure connections to the directory

After following the recommendations in this chapter, make sure that you test your installation to verify that it behaves as expected before putting the server into production.

Setting Up a System Account for a Server

Do not run DS servers as the system superuser (root). When applications run as superuser, the system effectively does not control their actions. When running the server as superuser, a bug in the server could affect other applications or the system itself.

After setting up a system account for the server, and using that account only to run the server, you can use system controls to limit user access.

The user running the server must have access to use the configured ports. Make sure you configure the system to let the user access privileged ports such as 389 and 636 if necessary. Make sure you configure the firewall to permit access to the server ports.

The user running the server must have access to all server files, including configuration files, data files, log files, keystores, truststores and their password files, and other files. By default, DS software

lets users in the same group as the user running the server read server files, though not directory data files.

The user running the server does not, however, need access to login from a remote system or to perform actions unrelated to the directory service.

Set up the user account to prevent other users from reading configuration files. On UNIX, set an appropriate umask such as `027` to prevent users in other groups from accessing server files. On Windows, use file access control to do the same. Do consider letting all users to run command-line tools. What a user can do with tools depends on server access control mechanisms. For details, see "Setting Appropriate File Permissions" in the *Security Guide*.

You can create a UNIX service script to start the server at system startup and stop the server at system shutdown by using the **create-rc-script** command. For details see "*create-rc-script — script to manage OpenDJ as a service on UNIX*" in the *Reference*.

You can use the **windows-service** command to register the DS server as a Windows service. For details, see "*windows-service — register DS as a Windows Service*" in the *Reference*.

Using Java Security Updates

Security updates are regularly released for the Java runtime environment.

Make sure that your operational plans provide for deploying Java security updates to systems where you run DS software.

When you set up DS servers, the path to the Java runtime environment is saved in the configuration. The server continues to use that Java version until you change the configuration as described below.

To Apply Java Security Updates

Follow these steps each time an update to the Java runtime environment changes the Java home path where the runtime environment is installed:

1. (Optional) If the previous Java update used an unlimited strength cryptography policy, install the policy for the updated Java runtime environment as described in "Using Unlimited Strength Cryptography" in the *Security Guide*.

2. (Optional) If the DS server relies on any CA certificates that were added to the Java runtime environment truststore, `$JAVA_HOME/Home/jre/lib/security/cacerts` (Java 8) or `$JAVA_HOME/Home/lib/security/cacerts` (Java 11), for the previous update, add them to the truststore for the update Java runtime environment.

3. Edit the `default.java-home` setting in the file `config/java.properties` to use the new path.

   The setting should reflect the updated Java home:

   ```
   default.java-home=/path/to/updated/java/jre
   ```

4. Restart the DS server to use the updated Java runtime environment:

```
$ stop-ds --restart
```

Only Enable Necessary Services

The setup process enables DS server connection handlers, enabling at least an administration connection handler. You can choose to disable other connection handlers after setting up the server. For example, if the setup process enables the (cleartext) LDAP connection handler, but only LDAPS or HTTPS is used, then set the LDAP connection handler property to `enabled:false` by using the **dsconfig set-connection-handler-prop** command.

Use the **status** command to check which connection handlers are enabled.

Configure Logging Appropriately

By default, DS servers write log messages to files when an error is encountered and when a server is accessed. Access logs tend to be much more intensively updated than error logs. You can also configure debug logging, generally too verbose for continuous use in production, and audit logging, which uses the access log mechanism to record changes. Debug and audit logs are not enabled by default. For details, see "Server Logs".

The default DS server error log levels and log rotation and retention policies are set to prevent the logs from harming performance or filling up the disk while still making it possible to perform basic troubleshooting. If you must set a more verbose error log level or if you must activate debug logging on a production system for more advanced troubleshooting, be aware that extra logging can negatively impact performance and generate large files on heavily used servers. When finished troubleshooting, reset the log configuration for more conservative logging.

The audit log for DS servers is not for security audits. Instead it records changes in LDIF. The audit log is intended to help you as server administrator to diagnose problems in the way applications change directory data. For change notification as a service, use the external change log instead. For details about the external change log, see "Change Notification For Your Applications".

Limit Use of the cn=Directory Manager Account

Directory manager accounts are stored in their own backends. In order to bootstrap the system, the default directory superuser, `cn=Directory Manager`, is not subject to access control and has privileges to perform almost every administrative operation, including changing privileges.

Use this account like you use the superuser (root) account on UNIX or the Administrator account on Windows: Use it only when you must.

Instead of allowing other applications to perform operations as the directory superuser, `cn=Directory Manager`, either create alternative administrators with limited privileges, or explicitly assign directory administrator rights to specific accounts.

When creating alternative administrators, consider limiting their privileges to prevent them from having `bypass-acl` and `privilege-change` privileges. For an example of how to do this, see "To Add Privileges for an Individual Administrator".

To explicitly assign rights to specific accounts, create a directory administrator group and add administrators as members. Use the group to assign privileges to the administrators. For details, see "To Add Privileges for a Group of Administrators". Create multiple administrator groups if necessary for your deployment.

In both cases, explicitly set up access control instructions (ACIs) to allow administrators to perform administrative actions. For details see "*Configuring Privileges and Access Control*". This prevents administrators from accidentally or intentionally overstepping their authority when managing DS servers and directory data, and you make it easier to audit what administrators can do.

Reconsider Default Global Access Control

Global ACIs or access policies are defined in the server configuration. Global access settings apply together with ACIs in the user data.

You can set up a server to apply the recommendations in this section by using the **setup** command option, `--productionMode`.

When you set up a server without using the `--productionMode` option, default global access control settings allow applications to:

- Read the root DSE
- Read server LDAP schema
- Read directory data anonymously
- Modify one's own entry
- Request extended operations and operations with certain controls

For details, see "Default Global ACIs".

If the default global access control settings do not match your requirements, make sure you change them on each server as the server configuration data is not replicated. Global ACIs have the same syntax as ACIs in the directory data. Global access policies are entries in the server configuration. For details about access control settings, see "*Configuring Privileges and Access Control*".

Default global access control settings can and often do change between releases. Review the release notes when upgrading to a new release. For details, see "*Compatibility*" in the *Release Notes*.

Generally it is appropriate to allow anonymous applications to read the root DSE, and to request the StartTLS extended operation over a cleartext connection, even if read access to most directory data requires authorization. The operational attributes on the root DSE indicate the server capabilities, allowing applications to discover interactively how to use the server. The StartTLS extended operation lets an application initiate a secure session starting on a port that does not require encryption.

Authenticated applications should be allowed to read schema operational attributes. LDAP schema operational attributes describe the data stored in the directory. An application that can read schema attributes and check that changes to directory data respect the LDAP schema before sending an update request.

To Minimize Global ACIs

Follow these steps to minimize global ACIs:

1. Remove existing global ACIs to prevent all access.

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --reset global-aci \
 --trustAll \
 --no-prompt
```

2. Allow limited global access for essential operations.

   This example allows the following limited access:

   - Authenticated users can request the ForgeRock Transaction ID control, which has OID `1.3.6.1.4.1.36733.2.1.5.1`.

     Other components in the ForgeRock platform use this control to share transaction IDs for common access logging.

     If you do not use common access logging, you can skip adding the global ACI for `Transaction ID control access`.

   - Anonymous users can request the Get Symmetric Key extended operation, which has OID `1.3.6.1.4.1.26027.1.6.3`, and the StartTLS extended operation, which has OID `1.3.6.1.4.1.1466.20037`.

     DS servers require Get Symmetric Key extended operation access to create and share secret keys for encryption.

     Directory client applications must be able to use the StartTLS operation to initiate a secure connection with an LDAP connection handler. This must be available to anonymous users

so that applications can initiate a secure connection before sending bind credentials to authenticate, for example.

If the directory deployment does not support StartTLS, then remove `|| 1.3.6.1.4.1.1466.20037` from the global ACI for `Anonymous extended operation access`.

- Anonymous and authenticated users can read information about the LDAP features that DS servers support according to the global ACI named `User-Visible Root DSE Operational Attributes`.

  This exposes metadata publicly for the following attributes:

  `namingContexts`

  The base DNs for user data

  `supportedAuthPasswordSchemes`

  Supported `authPassword` storage schemes for pre-encoded passwords

  `supportedControl`

  Supported LDAP controls by OID

  `supportedExtension`

  Supported LDAP extended operations by OID

  `supportedFeatures`

  Supported optional LDAP features by OID

  `supportedLDAPVersion`

  Supported LDAP versions

  `supportedSASLMechanisms`

  Supported SASL mechanisms

  `supportedTLSCiphers`

  Supported cipher suites for transport layer security

  `supportedTLSProtocols`

  Supported protocols for transport layer security

  `vendorName`

  Name of the LDAP server implementer

`vendorVersion`

Version of the LDAP server implementation

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"1.3.6.1.4.1.36733.2.1.5.1\")\
 (version 3.0; acl \"Transaction ID control access\"; allow(read)\
 userdn=\"ldap:///all\";)" \
 --add global-aci:"(extop=\"1.3.6.1.4.1.26027.1.6.3 || 1.3.6.1.4.1.1466.20037\") \
 (version 3.0; acl \"Anonymous extended operation access\";\
 allow(read) userdn=\"ldap:///anyone\";)" \
 --add global-aci:"(target=\"ldap:///\")(targetscope=\"base\")\
 (targetattr=\"objectClass||namingContexts||supportedAuthPasswordSchemes\
 ||supportedControl||supportedExtension||supportedFeatures||supportedLDAPVersion\
 ||supportedSASLMechanisms||supportedTLSCiphers||supportedTLSProtocols||\
 vendorName||vendorVersion\")(version 3.0;\
 acl \"User-Visible Root DSE Operational Attributes\";\
 allow (read,search,compare) userdn=\"ldap:///anyone\";)" \
 --add global-aci:"(target=\"ldap:///cn=schema\")(targetscope=\"base\")\
 (targetattr=\"objectClass||attributeTypes||dITContentRules||dITStructureRules\
 ||ldapSyntaxes||matchingRules||matchingRuleUse||nameForms||objectClasses\")\
 (version 3.0; acl \"User-Visible Schema Operational Attributes\";\
 allow (read,search,compare) userdn=\"ldap:///all\";)" \
 --trustAll \
 --no-prompt
```

3. Add or update global ACIs as required by known client applications.

   Work with your partners and with client application documentation for details.

   For example, ForgeRock Access Management servers might require access to update schema definitions under `cn=schema`, and to use the Persistent Search control.

   The following capabilities are useful with the REST to LDAP gateway:

   Access to Read Entry controls

   By default, the DS REST to LDAP gateway uses the Pre-Read (OID: `1.3.6.1.1.13.1`) and Post-Read (OID: `1.3.6.1.1.13.2`) controls to read an entry before it is deleted, or to read an entry after it is added or modified.

   To determine whether access to request the controls is required by the deployment, check the configuration for `readOnUpdatePolicy` as described in "Gateway REST2LDAP Configuration File" in the *Reference*.

Access to the Subtree Delete control

By default, the DS REST to LDAP gateway uses the LDAP Subtree Delete (OID: `1.2.840.113556.1.4.805`) control for delete operations.

To determine whether access to request the controls is required by the deployment, check the configuration for `useSubtreeDelete` as described in "Gateway REST2LDAP Configuration File" in the *Reference*.

Access to the Permissive Modify control

By default, the DS REST to LDAP gateway uses the LDAP Permissive Modify (OID: `1.2.840.113556.1.4.1413`) control for LDAP modify operations resulting from patch and update operations.

To determine whether access to request the controls is required by the deployment, check the configuration for `usePermissiveModify` as described in "Gateway REST2LDAP Configuration File" in the *Reference*.

The following example adds control access for a REST to LDAP gateway with a default configuration:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"1.3.6.1.1.13.1||1.3.6.1.1.13.2\
 ||1.2.840.113556.1.4.805||1.2.840.113556.1.4.1413\")\
 (version 3.0; acl \"REST to LDAP control access\"; allow(read) userdn=\"ldap:///all\";)" \
 --trustAll \
 --no-prompt
```

Protect Network Connections

Server protocols like LDAP, HTTP, JMX, and replication rely on transport layer security to protect network connections. For evaluation and initial testing, you might find it useful to inspect network messages without decrypting them. For final testing and production environments, secure the connections.

Transport layer security depends on public key infrastructure when negotiating encryption. DS servers have keystores for handling their key pairs and public key certificates. For details, see "*Changing Server Certificates*".

The DS server setup process can simplify installation by self-signing certificates for server key pairs. Self-signed certificates are not recognized by applications until you add them to the application's

truststore. This is not a problem when you control both the service and the applications. Self-signed certificates are generally fine even in production systems for administrative and replication connections not used by other applications. For connection handlers that primarily serve applications you do not control, have the server public key certificate signed by a well-known CA so that the applications can recognize the certificate by default. For details on setting up connection handlers for secure communications, see "*Configuring Connection Handlers*".

You can use an access control setting to require secure communications for most operations. Keep a global access control setting that allows anonymous access to request the StartTLS extended operation. When using ACIs, for all operations other than requesting StartTLS, use ACIs whose subject sets `authmethod = ssl`, and also sets `ssf` appropriately. When using global access control policies, use a `connection-minimum-ssf` setting that enforces use of transport layer security, such as 128 or 256.

A security strength factor (`ssf`) is set when the server negotiates connection security with a client application. The `ssf` setting in an ACI subject indicates acceptable security strength factors for the target operation. The server can then check whether the security strength factor for the connection is acceptable according to ACIs that apply. The `ssf` setting in an ACI takes an integer between 0 and 1024. `ssf = 0` (or not set) means cleartext is acceptable. `ssf = 1` calls for integrity protection, meaning the connection should prevent messages from being corrupted between the sender and the receiver. `ssf >= integer` where *integer* is two or more calls for integrity and confidentiality protection. Confidential messages are encrypted. Integers larger than one reflect the effective key size of the cipher negotiated between a server and the LDAP client application. With the `ssf` setting, the aim is to achieve a balance. If not set, or set too low, the server and client can negotiate a connection that is not secure. If set too high, the server and some clients might not be able to negotiate connection settings at all.

When a server and a client application negotiate connection security, they must agree on a security protocol and cipher suite. By default, a server supports all the SSL and TLS protocols and the cipher suites supported by the underlying Java virtual machine. The list can include protocols and ciphers that are not secure enough for the production environment. You can limit the security protocols and ciphers to those that are secure enough. For an example of how to change the settings for a connection handler, see "TLS Protocols and Cipher Suites". You can also change the settings on the administration connector with the **dsconfig set-administration-connector-prop** command, and change the settings for replication by changing the crypto manager settings with the **dsconfig set-crypto-manager-prop** command.

Use Appropriate Password Storage and Password Policies

Make sure you keep passwords secret in production. The server configuration includes files that hold passwords. Command-line tools allow users to provide password credentials. Passwords are also stored in directory data. This section looks at how to protect passwords in each situation.

Passwords in Configuration Files

DS servers store passwords in configuration files.

For replicated servers, the `admin-backend.ldif` file stores a password hash for the global administrator, such as `cn=admin,cn=Administrators,cn=admin data`. By default the password storage algorithm is Salted SHA512, a salted form of the 512-bit SHA-2 message digest algorithm. Permissions on the current copy of the file make it readable and writable only by the user running the server. Use a storage scheme that protects the password in the `admin-backend.ldif` file.

By default, DS servers store passwords for keystores and truststores in configuration files with `.pin` extensions. These files contain the cleartext, randomly generated passwords. Keep the PIN files readable and writable only by the user running the server. Alternatively, you can use the **dsconfig** command to configure the server to store keystore and truststore passwords in environment variables or Java properties if your procedures make these methods more secure in production. The settings to change are those of the Key Manager Providers and Trust Manager Providers.

Passwords as Command-Line Arguments

DS commands supply credentials for any operations that are not anonymous. Password credentials can be supplied as arguments such as the `--bindPassword` *`password`* option shown in many of the examples in the documentation. The passwords for keystores and truststores are handled in the same way. This is not recommended in production as the password appears in the command. Passwords can also be supplied interactively by using a `-` in the commands, as in `--bindPassword -`. The following example demonstrates a password supplied interactively:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword - \
 --baseDN "cn=Directory Manager" \
 "(&)" \
 userPassword
Password for user 'cn=Directory Manager':
dn: cn=Directory Manager
userPassword: {PBKDF2}10000:<hash>
```

Notice that the password appears neither in the shell history, nor in the terminal session.

When using scripts where the password cannot be supplied interactively, passwords can be read from files. For example, the `--bindPasswordFile` *`file`* option takes a file that should be readable only by the user running the command. It is also possible to set passwords in the `tools.properties` file for the user. This file is located in the user's home directory, on UNIX `~/.opendj/tools.properties`, and on Windows, typically `C:\Documents and Settings\`*`username`*`\.opendj\tools.properties`, though the location can depend on the Java runtime environment used. Here as well, make sure that the file is readable only by the user. Alternatively use other approaches that work with scripts such as Java properties or environment variables, depending on what method is most secure in production.

Passwords in Directory Data

DS servers encode users' passwords before storing them. A variety of built-in password storage schemes are available, using either one-way (hash) or reversible algorithms. The default storage schemes use one-way algorithms to make it computationally difficult to recover the cleartext password values even when given full access to the files containing stored password values.

For details see "Configuring Password Storage".

In DS servers, password policies govern password storage schemes, valid password values, password term duration, account lockout, and others. For example you can configure password policies that prevent users from setting weak passwords and from reusing passwords. DS software provides a wide range of alternatives. For details, see "*Configuring Password Policy*".

Protect DS Server Files

By default, DS servers do not encrypt server files or directory data. The only attribute values stored in encrypted or digest form are passwords. For instructions on encrypting entries and index content, see "Encrypting Directory Data". For instructions on encrypting change log content, see "To Encrypt External Change Log Data".

If you set up an appropriate user account for the server as described in "Setting Up a System Account for a Server", and unpacked the server files as that user, then the system should prevent other users from having overly permissive access to server files.

Included in the files that the server does not encrypt are LDIF exports of directory data. LDIF export files are readable and writable depending on the UNIX umask or Windows file access control settings for the user who runs the command to export the LDIF. The **export-ldif** command can compress the LDIF, but does not have an option for encrypting LDIF.

Directory backup archives can be encrypted, but are not encrypted by default. Backup archive file permissions depend on the UNIX umask or Windows file access control settings. When using the **backup** command, run an online backup and supply the `--encrypt` option as shown in the following example:

```
$ backup \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword - \
 --backupAll \
 --backupDirectory /path/to/opendj/bak \
 --encrypt \
 --start 0
Password for user 'cn=Directory Manager':
Backup task <datestamp> scheduled to start...
```

The server uses its Crypto Manager configuration to determine how to encrypt the backup archive data. The `--encrypt` option is not available for offline back up. If you back up server data offline, plan to protect the files separately.

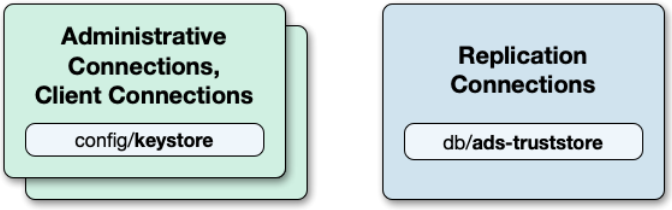
Chapter 22
Changing Server Certificates

This chapter covers how to renew certificates and replace server key pairs. In this chapter you will learn to:

- Renew an expiring certificate
- Replace a key pair for securing a connection handler
- Replace a key pair used for replication

DS servers use one keystore for key pairs to secure adminstration and client connections, and a separate keystore for replication, as shown in "Keystores and Truststores".

Keystores and Truststores



By default, DS servers use file-based keystores, `/path/to/opendj/config/keystore` and `/path/to/opendj/db/ads-truststore`:

- The `keystore` file holds keys for securing administrative and client connections.

  The key pair for the server has default alias `server-cert`.

  The cleartext password is stored in `keystore.pin`. This password serves as the password for the keystore and for any private keys it contains, including the `server-cert` key. The password for the keystore and for private keys must be the same. DS servers do not support using different passwords for the keystore and private keys.

  This is where you import trusted certificates for external applications.

- The `ads-truststore` file holds the server's key pair for securing replication connections, and other replicas' public key certificates.

The key pair for the server has default alias `ads-certificate`.

The cleartext password is stored in `ads-truststore.pin`. This password serves as the password for the keystore and for any private keys it contains, including the `ads-certificate` key. The password for the keystore and for private keys must be the same.

This keystore is synchronized with the certificates under the base DN `cn=admin data`. Do not change this keystore directly unless you understand the impact on the server configuration.

To Renew a Server Certificate

This procedure describes how to renew a certificate, for example, because the certificate is going to expire.

1. Get the certificate signed again using one of the following alternatives:

   - Self-sign the certificate.

     The following example self-signs the certificate with alias `server-cert`, setting the expiry to 7300 days (20 years) from the current time:

     ```
     $ keytool \
      -selfcert \
      -alias server-cert \
      -validity 7300 \
      -keystore /path/to/opendj/config/keystore \
      -storetype PKCS12 \
      -storepass:file /path/to/opendj/config/keystore.pin
     ```

   - Create a certificate signing request, have it signed by a CA, and import the signed certificate from the CA reply.

     For examples of the **keytool** commands to use, see "To Set Up a CA-Signed Certificate".

2. If client applications trust the self-signed certificate, have them import the renewed certificate.

   For details, see "To Import the Server Certificate".

To Replace a Server Key Pair

This procedure shows how to replace a server key pair. This procedure does not apply for replication key pairs. Instead, see "To Replace the Key Pair Used for Replication".

1. Record the alias of the existing key pair.

   This example shows a key pair that uses the default alias, `server-cert`:

```
$ keytool \
 -list \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin
...
server-cert, <date>, PrivateKeyEntry,
...
```

Connection handlers specify which key pair to use by specifying the alias in the `ssl-cert-nickname` property.

2. Generate a new key pair with a new alias.

   The following example generates a key pair that will have a self-signed certificate with alias `new-server-cert`:

```
$ keytool \
 -genkeypair \
 -keyalg RSA \
 -alias new-server-cert \
 -validity 7300 \
 -ext "san=dns:opendj.example.com" \
 -dname "CN=opendj.example.com, O=OpenDJ RSA Self-Signed Certificate" \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -keypass:file /path/to/opendj/config/keystore.pin \
 -storepass:file /path/to/opendj/config/keystore.pin
```

3. Get the new certificate signed, using one of the following alternatives:

   - Self-sign the certificate:

```
$ keytool \
 -selfcert \
 -alias new-server-cert \
 -validity 7300 \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin
```

   - Create a certificate signing request, have it signed by a CA, and import the signed certificate from the CA reply.

     For examples of the **keytool** commands to use, see "To Set Up a CA-Signed Certificate".

4. Restart the server to reload the keystore:

```
$ stop-ds --restart
```

5. For each connection handler referencing the alias of the old key pair, update the connection handler to use the new alias.

   The following example updates the LDAPS connection handler to use the new key pair:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name LDAPS \
 --set listen-port:1636 \
 --set enabled:true \
 --set use-ssl:true \
 --set ssl-cert-nickname:new-server-cert \
 --trustAll \
 --no-prompt
```

   Repeat this step for all connection handlers using the old key pair.

6. If you have client applications trusting the self-signed certificate, have them import the new one.

   For details, see "To Import the Server Certificate".

7. (Optional) Once the old key pair alias is no longer used anywhere, you can optionally delete the keys using the **keytool -delete** command with the old alias.

To Replace the Key Pair Used for Replication

Follow these steps to replace the key pair that is used to secure replication connections.

1. Generate a new key pair for the server.

   The changes you perform are replicated across the topology.

   The DS has an `ads-certificate` and private key, which is a local copy of the key pair used to secure replication connections.

   To generate the new key pair, you remove the `ads-certificate` key pair, prompt the server to generate a new `ads-certificate` key pair, and then add a copy to the administrative data using the MD5 fingerprint of the certificate to define the RDN.

   a. Delete the `ads-certificate` entry:

```
$ ldapmodify \
 --port 1389 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: ds-cfg-key-id=ads-certificate,cn=ads-truststore
changetype: delete

Processing DELETE request for ds-cfg-key-id=ads-certificate,cn=ads-truststore
DELETE operation successful for DN ds-cfg-key-id=ads-certificate,
 cn=ads-truststore
```

b. Prompt the server to generate a new, self-signed `ads-certificate` key pair.

You do this by adding an `ads-certificate` entry with object class `ds-cfg-self-signed-cert-request`:

```
$ ldapmodify \
 --port 1389 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: ds-cfg-key-id=ads-certificate,cn=ads-truststore
changetype: add
objectclass: ds-cfg-self-signed-cert-request

Processing ADD request for ds-cfg-key-id=ads-certificate,cn=ads-truststore
ADD operation successful for DN ds-cfg-key-id=ads-certificate,cn=ads-truststore
```

c. Retrieve the `ads-certificate` entry:

```
$ ldapsearch \
 --port 1389 \
 --hostname opendj.example.com \
 --baseDN cn=ads-truststore \
 "(ds-cfg-key-id=ads-certificate)"
dn: ds-cfg-key-id=ads-certificate,cn=ads-truststore
ds-cfg-key-id: ads-certificate
ds-cfg-public-key-certificate;binary:: MIIB6zCCAVSgAwIBAgIEDKSUFjANBgkqhkiG9w0BA
 QUFADA6MRswGQYDVQQKExJPcGVuREogQ2VydGlmaWNhdGUxGzAZBgNVBAMTEm9wZW5hbS5leGFtcGxl
 LmNvbTAeFw0xMzAyMDcxMDMwMzNaFw0zMzAyMDIxMDMwMzNaMDoxGzAZBgNVBAoTEk9wZW5ESiBDZXJ
 0aWZpY2F0ZTEbMBkGA1UEAxMSb3BlbmFtLmV4YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNAD
 CBiQKBgQCfGLAiUOz4sC8CM9T5DPTk9V9ErNC8N59XwBt1aN7UjhQl4/JZZsetubtUrZBLS9cRrnYdZ
 cpFgLQNEmXifS+PdZ0DJkaLNFmd8ZX0spX8++fb4SkkggkmNRmi1fccDQ/DHMlwl7kk884lXummrzcD
 GbZ7p4vnY7y7GmD1vZSP+wIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAJciUzUP8T8A9VV6dQB0SYCNG1o
 7IvpE7jGVZh6KvM0m5sBNX3wPbTVJQNij3TDm8nx6yhi6DUkpiAZfz/OBL5k+WSw80TjpIZ2+klhP1s
 srsST4Um4fHzDZXOXHR6NM83XxZBsR6MazYecL8CiGwnYW2AeBapzbAnGn1J831q1q
objectClass: top
objectClass: ds-cfg-instance-key
```

d. Retrieve the MD5 fingerprint of the `ads-certificate`:

- If you have only Java 11, export the `ads-certificate` to a binary, CRT-format file, and then get the MD5 fingerprint with the **openssl md5** command:

```
$ keytool \
 -exportcert \
 -alias ads-certificate \
 -file /path/to/ads-cert.crt \
 -keystore /path/to/opendj/db/ads-truststore/ads-truststore \
 -storepass:file /path/to/opendj/db/ads-truststore/ads-truststore.pin
Certificate stored in file </path/to/ads-cert.crt>
$ openssl md5 /path/to/ads-cert.crt
MD5(/path/to/ads-cert.crt)= 073580d8f3cee1399cd073db6cfacc1c
```

- If you have Java 8, display the fingerprint using the Java 8 **keytool** command:

```
$ keytool \
 -list \
 -v \
 -alias ads-certificate \
 -keystore /path/to/opendj/db/ads-truststore/ads-truststore \
 -storepass:file /path/to/opendj/db/ads-truststore/ads-truststore.pin
...
Certificate fingerprints:
  MD5:  07:35:80:D8:F3:CE:E1:39:9C:D0:73:DB:6C:FA:CC:1C
```

e. Using the MD5 fingerprint and the certificate entry, prepare LDIF to update `cn=admin data` with the new server certificate:

```
$ cat /path/to/update-server-cert.ldif
dn: ds-cfg-key-id=073580D8F3CEE1399CD073DB6CFACC1C,cn=instance keys,
 cn=admin data
changetype: add
ds-cfg-key-id: 073580D8F3CEE1399CD073DB6CFACC1C
ds-cfg-public-key-certificate;binary:: MIIB6zCCAVSgAwIBAgIEDKSUFjANBgkqhkiG9w0BA
 QUFADA6MRswGQYDVQQKExJPcGVuREogQ2VydGlmaWNhdGUxGzAZBgNVBAMTEm9wZW5hbS5leGFtcGxl
 LmNvbTAeFw0xMzAyMDcxMDMwMzNaFw0zMzAyMDIxMDMwMzNaMDoxGzAZBgNVBAoTEk9wZW5ESiBDZXJ
 0aWZpY2F0ZTEbMBkGA1UEAxMSb3BlbmFtLmV4YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNAD
 CBiQKBgQCfGLAiUOz4sC8CM9T5DPTk9V9ErNC8N59XwBt1aN7UjhQl4/JZZsetubtUrZBLS9cRrnYdZ
 cpFgLQNEmXifS+PdZ0DJkaLNFmd8ZX0spX8++fb4SkkggkmNRmi1fccDQ/DHMlwl7kk884lXummrzcD
 GbZ7p4vnY7y7GmD1vZSP+wIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAJciUzUP8T8A9VV6dQB0SYCNG1o
 7IvpE7jGVZh6KvM0m5sBNX3wPbTVJQNij3TDm8nx6yhi6DUkpiAZfz/OBL5k+WSw80TjpIZ2+klhP1s
 srsST4Um4fHzDZXOXHR6NM83XxZBsR6MazYecL8CiGwnYW2AeBapzbAnGn1J831q1q
objectClass: top
objectClass: ds-cfg-instance-key

dn: cn=opendj.example.com:4444,cn=Servers,cn=admin data
changetype: modify
replace: ds-cfg-key-id
ds-cfg-key-id: 073580D8F3CEE1399CD073DB6CFACC1C
```

f. Update the administrative data, causing the server to create a copy of the new `ads-certificate` with its MD5 signature as the alias in the `ads-truststore`:

```
$ ldapmodify \
 --port 1389 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 /path/to/update-server-cert.ldif
Processing ADD request for ds-cfg-key-id=073580D8F3CEE1399CD073DB6CFACC1C,
 cn=instance keys,cn=admin data
ADD operation successful for DN ds-cfg-key-id=073580D8F3CEE1399CD073DB6CFACC1C,
 cn=instance keys,cn=admin data
Processing MODIFY request for cn=opendj.example.com:4444,cn=Servers,
 cn=admin data
MODIFY operation successful for DN cn=opendj.example.com:4444,cn=Servers,
 cn=admin data
```

2. Force the server to reopen replication connections using the new key pair.

   Stop replication temporarily and then start it again as described in "To Stop Replication Temporarily For a Replica":

```
$ dsconfig \
 set-synchronization-provider-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set enabled:false \
 --no-prompt

$ dsconfig \
 set-synchronization-provider-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Multimaster Synchronization" \
 --set enabled:true \
 --no-prompt
```

Chapter 23
Moving Servers

This chapter explains how to move DS servers, which you must do if you change the hostname, the filesystem layout, or the system where the server is installed. In this chapter you will learn to:

- Prepare for the move, especially when the server is replicated, and when the directory service remains available during the move
- Perform the configuration needed to move the server

When you change where a server is deployed, you must take host names, port numbers, and certificates into account. The changes can also affect your replication configuration.

Overview

From time to time you might change server hardware, file system layout, or host names. At those times you move the services running on the system. You can move DS data between servers and operating systems. Most of the configuration is also portable.

Two aspects of the configuration are not portable:

1. Server certificates contain the host name of the system. Even if you did not set up secure communications when you installed the server, the server still has a certificate used for secure communications on the administrative port.

   To resolve the issue with server certificates, you can change the server certificates during the move as described in this chapter.

2. Replication configuration includes the host name and administrative port numbers.

   You can work around the issue with replication configuration by unconfiguring replication for the server before the move, and then configuring and initializing replication again after the move.

Before You Move

Take a moment to determine whether you find it quicker and easier to move your server, or to recreate a copy. To recreate a copy, install a new server, set up the new server configuration

to match the old, and then copy only the data from the old server to the new server, initializing replication from existing data, or even from LDIF if your database is not too large.

After you decide to move a server, start by taking it out of service. Taking it out of service means directing client applications elsewhere, and then preventing updates from client applications, and finally unconfiguring replication. Directing client applications elsewhere depends on your network configuration and possibly on your client application configuration. The other two steps can be completed with the **dsconfig** and **dsreplication** commands.

To Take the Server Out of Service

1. Direct client applications to other servers.

   How you do this depends on your network and client application configurations.

2. Prevent the server from accepting updates from client applications:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set writability-mode:internal-only \
 --trustAll \
 --no-prompt
```

3. Unconfigure replication for the server:

```
$ dsreplication \
 unconfigure \
 --unconfigureAll \
 --hostname replica.example.com \
 --port 4444 \
 --adminUID admin \
 --adminPassword password \
 --trustAll \
 --no-prompt
```

4. With the server no longer receiving traffic or accepting updates from clients, and no longer replicating to other servers, you can shut it down in preparation for the move:

```
$ stop-ds --quiet
```

5. (Optional) You might also choose to remove extra log files from the server `logs/` directory before moving the server.

Moving a Server

Now that you have decided to move your server, and prepared for the move, you must not only move the files but also fix the configuration and the server certificates, and then configure replication.

To Move the Server

1. Move the contents of the server installation directory to the new location.
2. (Optional) If you must change port numbers, edit the port numbers in `config/config.ldif`, carefully avoiding changing any whitespace or other lines in the file.
3. Change server certificates as described in "*Changing Server Certificates*".
4. Start the server:

```
$ start-ds --quiet
```

5. Configure and initialize replication:

```
$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 opendj.example.com \
 --port1 4444 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --replicationPort1 8989 \
 --host2 replica.example.com \
 --port2 4444 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --replicationPort2 8989 \
 --trustAll \
 --no-prompt

$ dsreplication \
 pre-external-initialization \
 --hostname replica.example.com \
 --port 4444 \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --trustAll \
 --no-prompt

$ dsreplication \
 post-external-initialization \
 --hostname replica.example.com \
 --port 4444 \
```

```
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --trustAll \
 --no-prompt
```

6. Accept updates from client applications:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname replica.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set writability-mode:enabled \
 --trustAll \
 --no-prompt
```

7. Direct client applications to the server.

Chapter 24

Troubleshooting Server Problems

This chapter describes how to troubleshoot common server problems, and how to collect information necessary when seeking support help. In this chapter you will learn to:

- Identify server problems systematically as a first troubleshooting step
- Troubleshoot problems with installation and upgrade procedures, directory data import, data replication, and secure connections
- Reset lost administrator passwords
- Enable debug logging judiciously when solving problems
- Prevent applications from accessing the server when solving problems
- Troubleshoot problems with the way client applications access the directory
- Prepare evidence when asking a directory expert for help

Identifying the Problem

In order to solve your problem methodically, save time by defining the problem clearly up front. In a replicated environment with multiple DS servers and many client applications, it can be particularly important to pin down not only the problem (difference in observed behavior compared to expected behavior), but also the circumstances and steps that lead to the problem occurring.

Answer the following questions:

- How do you reproduce the problem?
- What exactly is the problem? In other words, what is the behavior you expected? What is the behavior you observed?
- When did the problem start occurring? Under similar circumstances, when does the problem not occur?
- Is the problem permanent? Intermittent? Is it getting worse? Getting better? Staying the same?

Pinpointing the problem can sometimes indicate where you should start looking for solutions.

Troubleshooting Installation and Upgrade

Installation and upgrade procedures result in a log file tracing the operation. The log location differs by operating system, but look for lines in the command output of the following form:

```
See file for a detailed log of this operation.
```

Prevent antivirus and intrusion detection systems from interfering with DS software.

Before using DS software with antivirus or intrusion detection software, consider the following potential problems:

Interference with normal file access

Antivirus and intrusion detection systems that perform virus scanning, sweep scanning, or deep file inspection are not compatible with DS file access, particularly database file access.

Antivirus and intrusion detection software can interfere with the normal process of opening and closing database working files. They may incorrectly mark such files as suspect to infection due to normal database processing, which involves opening and closing files in line with the database's internal logic.

Prevent antivirus and intrusion detection systems from scanning database and changelog database files.

At minimum, configure antivirus software to whitelist the DS server database files. By default, exclude the following file system directories from virus scanning:

- `/path/to/opendj/changelogDb/` (if replication is enabled)

  Prevent the antivirus software from scanning these changelog database files.

- `/path/to/opendj/db/`

  Prevent the antivirus software from scanning database files, especially `*.jdb` files.

Port blocking

Antivirus and intrusion detection software can block ports that DS uses to provide directory services.

Make sure that your software does not block the ports that DS software uses. For details, see "Limiting System and Administrative Access" in the *Security Guide*.

Negative performance impact

Antivirus software consumes system resources, reducing resources available to other services including DS servers.

Running antivirus software can therefore have a significant negative impact on DS server performance. Make sure that you test and account for the performance impact of running antivirus software before deploying DS software on the same systems.

When starting a directory server on a Linux system, make sure the server user can watch enough files. For details, see "Setting Maximum Inotify Watches" in the *Installation Guide*. If the server user cannot watch enough files, you might see an error message in the server log including the following text:

```
InitializationException: The database environment could not be opened:
 com.sleepycat.je.EnvironmentFailureException: (JE 7.5.11) /path/to/opendj/db/userRoot
 or its sub-directories to WatchService.
 UNEXPECTED_EXCEPTION: Unexpected internal Exception, may have side effects.
 Environment is invalid and must be closed.
```

Resetting Administrator Passwords

This section describes what to do if you forgot the password for Directory Manager or for the global (replication) administrator.

Resetting the Directory Manager's Password

By default, DS servers store the entry for Directory Manager in an LDIF backend. You must edit the file in order to reset the Directory Manager's password as shown in the following steps:

1. Generate the encoded version of the new password using the DS **encode-password** command:

   ```
   $ encode-password --storageScheme SSHA512 --clearPassword password
   encode-password --storageScheme SSHA512 --clearPassword password
   ```

2. Stop the server while you edit the LDIF file for the backend:

   ```
   $ stop-ds --quiet
   ```

3. Replace the existing password with the password you generated.

   By default, the LDIF file for the directory superuser's entry is `db/rootUser/rootUser.ldif`. In the file, carefully replace the `userpassword` attribute value with the encoded version of the new password, taking care not to leave any whitespace at the end of the line:

   ```
   dn: cn=Directory Manager
   objectClass: top
   objectClass: person
   objectClass: organizationalPerson
   objectClass: inetOrgPerson
   givenName: Directory
   sn: Manager
   ds-rlim-size-limit: 0
   ds-rlim-time-limit: 0
   ds-rlim-idle-time-limit: 0
   ds-rlim-lookthrough-limit: 0
   ```

```
ds-rlim-cursor-entry-limit: 100000
ds-pwp-password-policy-dn: cn=Root Password Policy,cn=Password Policies,cn=config
ds-privilege-name: bypass-lockdown
ds-privilege-name: bypass-acl
ds-privilege-name: modify-acl
ds-privilege-name: config-read
ds-privilege-name: config-write
ds-privilege-name: ldif-import
ds-privilege-name: ldif-export
ds-privilege-name: backend-backup
ds-privilege-name: backend-restore
ds-privilege-name: server-lockdown
ds-privilege-name: server-shutdown
ds-privilege-name: server-restart
ds-privilege-name: disconnect-client
ds-privilege-name: cancel-request
ds-privilege-name: password-reset
ds-privilege-name: update-schema
ds-privilege-name: privilege-change
ds-privilege-name: unindexed-search
ds-privilege-name: subentry-write
ds-privilege-name: changelog-read
cn: Directory Manager
userPassword: <encoded-password>
```

4. Start the server again:

```
$ start-ds --quiet
```

5. Verify that you can administer the server as Directory Manager using the new password:

```
$ status \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 --hostname opendj.example.com \
 --port 4444 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --script-friendly
"isRunning" : true,
...
```

To Reset the Global Administrator's Password

When you configure replication, part of the process involves creating a global administrator and setting that user's password. This user is present on all replicas. If you chose default values, this user has DN `cn=admin,cn=Administrators,cn=admin data`. You reset the password as you would for any other user, though you do so as Directory Manager.

1. Use the **ldappasswordmodify** command to reset the global administrator's password:

```
$ ldappasswordmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --useStartTLS \
 --authzID "cn=admin,cn=Administrators,cn=admin data" \
 --newPassword password
```

2. Let replication copy the password change to other replicas.

Enabling Debug Logging

DS servers can write debug information and stack traces to the server debug log. What is logged depends both on debug targets that you create, and on the debug level that you choose.

To Configure Debug Logging

1. Create a debug target or targets.

   No debug targets are enabled by default:

```
$ dsconfig \
 list-debug-targets \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --trustAll \
 --no-prompt
Debug Target : enabled : debug-exceptions-only
-------------:---------:----------------------
```

   A debug target specifies a fully qualified DS Java package, class, or method for which to log debug messages at the level you specify:

```
$ dsconfig \
 create-debug-target \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --type generic \
 --target-name org.opends.server.api \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

2. Enable the debug log, `opendj/logs/debug`, which is not enabled by default:

```
    $ dsconfig \
 set-log-publisher-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --publisher-name "File-Based Debug Logger" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

   It is not necessary to restart the server. The server immediately begins to write debug messages to the log file.

3. Read messages in the debug log file:

```
$ tail -f /path/to/opendj/logs/debug
```

Caution

DS servers can generate a high volume of debug output. Use debug logging very sparingly on production systems.

Preventing Access While Fixing Issues

Misconfiguration can potentially put the DS server in a state where you must intervene, and where you need to prevent users and applications from accessing the directory until you are done fixing the problem.

DS servers provide a *lockdown mode* that allows connections only on the loopback address, and allows only operations requested by superusers, such as `cn=Directory Manager`. You can use lockdown mode to prevent all but administrative access while you repair a server.

To put the DS server into lockdown mode, the server must be running. You cause the server to enter lockdown mode by using a task. Notice that the modify operation is performed over the loopback address (accessing the DS server on the local host):

```
$ cat enter-lockdown.ldif
dn: ds-task-id=Enter Lockdown Mode,cn=Scheduled Tasks,cn=tasks
objectClass: top
objectClass: ds-task
ds-task-id: Enter Lockdown Mode
ds-task-class-name: org.opends.server.tasks.EnterLockdownModeTask

$ ldapmodify \
 --hostname localhost \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 enter-lockdown.ldif
```

The DS server logs a notice message in `logs/errors` when lockdown mode takes effect:

```
...msg=Lockdown task Enter Lockdown Mode finished execution
```

Client applications that request operations get a message concerning lockdown mode:

```
$ ldapsearch --port 1389 --baseDN "" --searchScope base "(objectclass=*)" +
The LDAP search request failed: 53 (Unwilling to Perform)
Additional Information:  Rejecting the requested operation because the server
is in lockdown mode and will only accept requests from root users over
loopback connections
```

You also leave lockdown mode by using a task:

```
$ cat leave-lockdown.ldif
dn: ds-task-id=Leave Lockdown Mode,cn=Scheduled Tasks,cn=tasks
objectClass: top
objectClass: ds-task
ds-task-id: Leave Lockdown Mode
ds-task-class-name: org.opends.server.tasks.LeaveLockdownModeTask

$ ldapmodify \
 --hostname localhost \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 leave-lockdown.ldif
```

The DS server logs a notice message when leaving lockdown mode:

```
...msg=Leave Lockdown task Leave Lockdown Mode finished execution
```

Troubleshooting LDIF Import

By default, DS directory servers require that LDIF data respect standards. In particular, a directory server checks that entries you import match the schema defined for the server. You can temporarily bypass this check using the `--skipSchemaValidation` option with the **import-ldif** command.

Also by default, DS servers ensure that entries have only one structural object class. You can relax this behavior by using the advanced global configuration property, `single-structural-objectclass-behavior`. This can be useful when importing data exported from Sun Directory Server. For example, to warn when entries have more than one structural object class instead of reject such entries being added, set `single-structural-objectclass-behavior:warn` as follows:

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --set single-structural-objectclass-behavior:warn \
 --trustAll \
 --no-prompt
```

By default, DS servers also check syntax for a number of attribute types. Relax this behavior using the advanced global configuration property, `invalid-attribute-syntax-behavior`.

When running **import-ldif**, you can use the `-R` *`rejectFile`* option to capture entries that could not be imported, and the `--countRejects` option to return the number of rejected entries as the **import-ldif** exit code.

Once you work through the issues with your LDIF data, reinstate the default behavior to ensure automated checking.

Troubleshooting TLS/SSL Connections

This section offers tips on troubleshooting LDAPS connections and connections set up with the StartTLS operation.

In order to trust the server certificate, client applications usually compare the signature on certificates with those of the Certificate Authorities (CAs) whose certificates are distributed with the client software. For example, the Java environment is distributed with a keystore holding many CA certificates:

```
# Java 11
$ keytool -list -cacerts -storepass changeit
# Java 8
$ keytool -list -keystore $JAVA_HOME/jre/lib/security/cacerts -storepass changeit
```

Self-signed server certificates generated during the setup process are not recognized as being signed by any CAs. Your software is configured not to trust the self-signed certificates by default. You must either configure the client applications to accept the self-signed certificates, or else use certificates signed by recognized CAs.

You can further debug the network traffic by collecting debug traces. To see the traffic going over TLS/SSL in debug mode, configure the DS server to dump debug traces from `javax.net.debug` into the `logs/server.out` file:

```
$ OPENDJ_JAVA_ARGS="-Djavax.net.debug=all" start-ds
```

Troubleshooting Certificates and SSL Authentication

Replication uses SSL to protect directory data on the network. In some configurations, replica can fail to connect to each other due to SSL handshake errors. This leads to error log messages such as the following:

```
...msg=SSL connection attempt from myserver (<ip>) failed:
Remote host closed connection during handshake
```

Notice these problem characteristics in the message above:

- The host name, `myserver`, is not fully qualified.

  You should not see non-fully qualified host names in the error logs. Non-fully qualified host names are a sign that the DS server has not been configured properly.

  Always install and configure DS servers using fully qualified host names. The DS administration connector, which is used by the **dsconfig** command, and also replication depend upon SSL and, more specifically, self-signed certificates for establishing SSL connections. If the host name used for connection establishment does not correspond to the host name stored in the SSL certificate then the SSL handshake can fail. For the purposes of establishing the SSL connection, a host name like `myserver` does not match `myserver.example.com`, and vice versa.

- The connection succeeded, but the SSL handshake failed, suggesting a problem with authentication or with the cipher or protocol negotiation. As most deployments use the same Java Virtual Machine (JVM), and the same JVM configuration for each replica, the problem is likely not related to SSL cipher or protocol negotiation, but instead lies with authentication.

Follow these steps on each DS server to check whether the problem lies with the host name configuration:

1. Make sure each DS server uses only fully qualified host names in the replication configuration. You can obtain a quick summary by running the following command against each server's configuration:

   ```
   $ grep ds-cfg-replication-server: config/config.ldif | sort | uniq
   ```

2. Make sure that the host names in DS certificates also contain fully qualified host names, and correspond to the host names found in the previous step:

```
# Examine the certificates used for the administration connector.
$ keytool -list -v -keystore config/keystore -storetype PKCS12 \
 -storepass:file config/keystore.pin | grep "^Owner:"

# Examine the certificates used for replication.
$ keytool -list -v -keystore db/ads-truststore/ads-truststore \
 -storepass:file db/ads-truststore/ads-truststore.pin | grep "^Owner:"
```

Sample output for a server on host `opendj.example.com` follows:

```
$ grep ds-cfg-replication-server: config/config.ldif |sort | uniq
ds-cfg-replication-server: opendj.example.com:8989
ds-cfg-replication-server: opendj.example.com:9989

$ keytool -list -v -keystore config/keystore -storetype PKCS12 \
-storepass:file config/keystore.pin | grep "^Owner:"
Owner: CN=opendj.example.com, O=Administration Connector Self-Signed Certificate

$ keytool -list -v -keystore db/ads-truststore/ads-truststore \
 -storepass:file db/ads-truststore/ads-truststore.pin | grep "^Owner:"
Owner: CN=opendj.example.com, O=DS Certificate
Owner: CN=opendj.example.com, O=DS Certificate
Owner: CN=opendj.example.com, O=DS Certificate
```

Unfortunately there is no easy solution to badly configured host names. It is often easier and quicker simply to reinstall your DS servers remembering to use fully qualified host names everywhere. Consider the following:

- When using the **setup** tool to install and configure a server ensure that the `-h` option is included, and that it specifies the fully qualified host name. Make sure you include this option even if you are not enabling SSL/StartTLS LDAP connections.

  If you are using the GUI installer, then make sure you specify the fully qualified host name on the first page of the wizard.

- When using the **dsreplication** tool to enable replication make sure that any `--host` options include the fully qualified host name.

If you cannot reinstall the server, follow these steps:

1. Unconfigure replication for each replica:

```
$ dsreplication \
 unconfigure \
 --unconfigureAll \
 --port adminPort \
 --hostname hostName \
 --adminUID admin \
 --adminPassword password \
 --trustAll \
 --no-prompt
```

2. Stop and restart each server in order to clear the in-memory ADS truststore backend.

3. Configure replication making certain that fully qualified host names are used throughout:

```
$ dsreplication \
 configure \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --host1 hostName1 \
 --port1 adminPort1 \
 --bindDN1 "cn=Directory Manager" \
 --bindPassword1 password \
 --replicationPort1 replPort1 \
 --host2 hostName2 \
 --port2 adminPort2 \
 --bindDN2 "cn=Directory Manager" \
 --bindPassword2 password \
 --replicationPort2 replPort2 \
 --trustAll \
 --no-prompt
```

4. Repeat the previous step for each remaining replica. In other words, host1 with host2, host1 with host3, host1 with host4, ..., host1 with hostN.

5. Initialize all remaining replica with the data from host1:

```
$ dsreplication \
 initialize-all \
 --adminUID admin \
 --adminPassword password \
 --baseDN dc=example,dc=com \
 --hostname hostName1 \
 --port 4444 \
 --trustAll \
 --no-prompt
```

6. Check that the host names are correct in the configuration and in the keystores by following the steps you used to check for host name problems. The only broken host name remaining should be in the key and truststores for the administration connector:

```
$ keytool -list -v -keystore config/keystore -storetype PKCS12 \
 -storepass:file config/keystore.pin |grep "^Owner:"
```

7. Stop each server, and then fix the remaining admin connector certificate as described in "To Replace a Server Key Pair".

Handling Compromised Keys

As explained in "*Changing Server Certificates*", DS servers have different keys for different purposes. The public keys used for replication are also used to encrypt shared secret symmetric keys, for example, to encrypt and to sign backups. This section looks at what to do if either a key pair or secret key is compromised.

How you handle the problem depends on which key was compromised:

- For a key pair used for a client connection handler and with a certificate signed by a certificate authority (CA), contact the CA for help. The CA might choose to publish a certificate revocation list (CRL) that identifies the certificate of the compromised key pair.

  Also make sure you replace the key pair. See "To Replace a Server Key Pair" for specific steps.

- For a key pair used for a client connection handler and that has a self-signed certificate, follow the steps in "To Replace a Server Key Pair", and make sure the clients remove the compromised certificate from their truststores, updating those truststores with the new certificate.

- For a key pair that is used for replication, mark the key as compromised as described below, and replace the key pair. See "To Replace the Key Pair Used for Replication" for specific steps.

  To mark the key pair as compromised, follow these steps:

  1. Identify the key entry by searching administrative data on the server whose key was compromised.

     The server in this example is installed on `opendj.example.com` with administration port `4444`:

     ```
     $ ldapsearch \
      --port 1389 \
      --hostname opendj.example.com \
      --baseDN "cn=admin data" \
      "(cn=opendj.example.com:4444)" ds-cfg-key-id
     dn: cn=opendj.example.com:4444,cn=Servers,cn=admin data
     ds-cfg-key-id: 4F2F97979A7C05162CF64C9F73AF66ED
     ```

     The key ID, `4F2F97979A7C05162CF64C9F73AF66ED`, is the RDN of the key entry.

  2. Mark the key as compromised by adding the attribute, `ds-cfg-key-compromised-time`, to the key entry.

The attribute has generalized time syntax, and so takes as its value the time at which the key was compromised expressed in generalized time. In the following example, the key pair was compromised at 8:34 AM UTC on March 21, 2017:

```
$ ldapmodify \
 --port 1389 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: ds-cfg-key-id=4F2F97979A7C05162CF64C9F73AF66ED,cn=instance keys,cn=admin data
changetype: modify
add: ds-cfg-key-compromised-time
ds-cfg-key-compromised-time: 201703210834Z
```

3. If the server uses encrypted or signed data, then the shared secret keys used for encryption or signing and associated with the compromised key pair should also be considered compromised. Therefore, mark all shared secret keys encrypted with the instance key as compromised.

   To identify the shared secret keys, find the list of secret keys in the administrative data whose `ds-cfg-symmetric-key` starts with the key ID of the compromised key:

   ```
   $ ldapsearch \
    --port 1389 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --baseDN "cn=secret keys,cn=admin data" \
    "(ds-cfg-symmetric-key=4F2F97979A7C05162CF64C9F73AF66ED*)" dn
   dn: ds-cfg-key-id=fba16e59-2ce1-4619-96e7-8caf33f916c8,cn=secret keys,cn=admin d
    ata

   dn: ds-cfg-key-id=57bd8b8b-9cc6-4a29-b42f-fb7a9e48d713,cn=secret keys,cn=admin d
    ata

   dn: ds-cfg-key-id=f05e2e6a-5c4b-44d0-b2e8-67a36d304f3a,cn=secret keys,cn=admin d
    ata
   ```

   For each such key, mark the entry with `ds-cfg-key-compromised-time` as shown above for the instance key.

Changes to administration data are replicated to other DS servers in the replication topology.

- For a shared secret key used for data encryption that has been compromised, mark the key entry with `ds-cfg-key-compromised-time` as shown in the example above that demonstrates marking the instance key as compromised.

  Again, changes to administration data are replicated to other DS servers in the replication topology.

Troubleshooting Client Operations

By default, DS servers log information about all LDAP client operations in `logs/logs/ldap-access.audit.json`. The following lines show the access log for a search operation with the filter `"(uid=bjensen)"`. The lines of JSON are formatted for readability:

```
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": <clientPort>
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "CONNECT",
    "connId": 0
  },
  "transactionId": "0",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 0,
    "elapsedTimeUnits": "MILLISECONDS"
  },
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
}
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": <clientPort>
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "SEARCH",
    "connId": 0,
    "msgId": 1,
    "dn": "dc=example,dc=com",
    "scope": "sub",
    "filter": "(uid=bjensen)",
    "attrs": ["ALL"]
  },
  "transactionId": "0",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 9,
    "elapsedTimeUnits": "MILLISECONDS",
```

```
    "nentries": 1
  },
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
}
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": <clientPort>
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "UNBIND",
    "connId": 0,
    "msgId": 2
  },
  "transactionId": "0",
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
}
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "<clientIp>",
    "port": <clientPort>
  },
  "server": {
    "ip": "<clientIp>",
    "port": 1389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "DISCONNECT",
    "connId": 0
  },
  "transactionId": "0",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 0,
    "elapsedTimeUnits": "MILLISECONDS",
    "reason": "Client Unbind"
  },
  "timestamp": "<timestamp>",
  "_id": "<uuid>"
}
```

A message corresponds to each client connection LDAP operation. The messages include information about what operation was performed, which client requested the operation, when it was completed, and more.

When HTTP access logging is also enabled, the server does not log the internal LDAP operations corresponding to HTTP requests by default. If you want to match HTTP client operations with internal LDAP operations, prevent the DS server from suppressing internal operations by using the **dsconfig** command to set the `suppress-internal-operations` advanced property to `false` for the LDAP logger. Match the values of the `request/connId` field in the HTTP access log with the same field in the LDAP access log.

To help diagnose client errors due to access permissions, see "Viewing Effective Rights" instead.

Clients Need Simple Paged Results Control

For some versions of Linux you might see a message in the DS access logs such as the following:

```
The request control with Object Identifier (OID) "1.2.840.113556.1.4.319"
cannot be used due to insufficient access rights
```

This message means clients are trying to use the simple paged results control without authenticating. By default, a global ACI allows only authenticated users to use the control.

To grant anonymous (unauthenticated) user access to the control, add a global ACI for anonymous use of the simple paged results control:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword "password" \
 --add global-aci:"(targetcontrol=\"1.2.840.113556.1.4.319\") \
 (version 3.0; acl \"Anonymous simple paged results access\"; allow(read) \
 userdn=\"ldap:///anyone\";)" \
 --trustAll \
 --no-prompt
```

Alternatively, stop the server, edit the `Anonymous control access` ACI carefully in `/path/to/opendj/config/config.ldif`, and restart the server. Unlike the **dsconfig** command, the `config.ldif` file is not a public interface, so this alternative should not be used for production servers.

Troubleshooting Replication

Replication can generally recover from conflicts and transient issues. Replication does, however, require that update operations be copied from DS server to DS server. It is therefore possible to experience temporary delays while replicas converge, especially when the write operation load is heavy. Tolerance for temporary divergence between replicas is what allows DS directory services to remain available to serve client applications even when networks linking the replicas go down.

In other words, the fact that DS directory services are loosely convergent rather than transactional is a feature, not a bug.

That said, you may encounter errors. Replication uses its own error log file, `logs/replication`. Error messages in the log file have `category=SYNC`. The messages have the following form. Here the line is folded for readability:

```
...msg=Replication server accepted a connection from 10.10.0.10/10.10.0.10:52859
 to local address 0.0.0.0/0.0.0.0:8989 but the SSL handshake failed.
 This is probably benign, but may indicate a transient network outage
 or a misconfigured client application connecting to this replication server.
 The error was: Remote host closed connection during handshake
```

DS servers maintain historical information about changes in order to bring replicas up to date, and to resolve replication conflicts. To prevent historical information from growing without limit, servers purge historical information after a configurable delay (`replication-purge-delay`, default: 3 days). A replica can become irrevocably out of sync if you restore it from a backup archive older than the purge delay, or if you stop it for longer than the purge delay. If this happens, unconfigure the replica, and then reinitialize it from a recent backup or from a server that is up to date.

Asking For Help

When you cannot resolve a problem yourself, and want to ask for help, clearly identify the problem and how you reproduce it, and the version of the server where the problem occurs. The version includes at least a version number and a build date stamp:

```
$ status --offline --version
ForgeRock Directory Services 6.5.6
Build <datestamp>
...
```

Be prepared to provide the following additional information:

- The Java home set in `config/java.properties`.
- Access and error logs showing what the server was doing when the problem started occurring.
- A copy of the server configuration file, `config/config.ldif`, in use when the problem started occurring.
- Other relevant logs or output, such as those from client applications experiencing the problem.
- A description of the environment where the server is running, including system characteristics, host names, IP addresses, Java versions, storage characteristics, and network characteristics. This helps to understand the logs, and other information.
- The `.zip` file generated using the **supportextract** command.

  For an example showing how to use the command, see "Examples" in the *Reference*.

Appendix A. On Using a Load Balancer

A load balancer might seem like a natural component for a highly available architecture. Directory services are highly available by design, however. When used with directory services, a load balancer can do more harm than good.

The Problem With Load Balancers

DS servers rely on data replication for high availability with tolerance for network partitions. The directory service continues to allow both read and write operations when the network is down. As a trade off, replication provides *eventual consistency*, not immediate consistency.

A load balancer configured to distribute connections or requests equitably across multiple servers can therefore *cause an application to get an inconsistent view of the directory data*. This problem arises in particular when a client application uses a pool of connections to access a directory service:

1. The load balancer directs a write request from the client application to a first server.

   The write request results in a change to directory data.

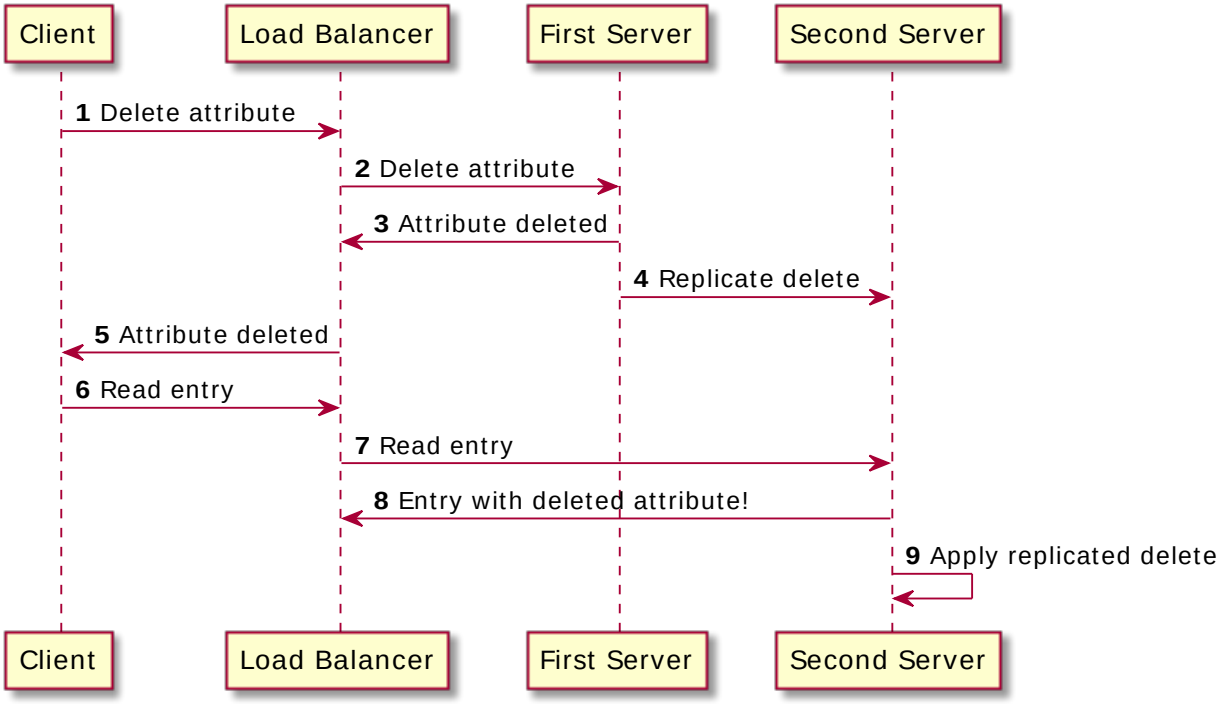
   The first server replicates the change to a second server, but replication is not instantaneous.

2. The load balancer directs a subsequent read request from the client application to a second server.

   The read request arrives before the change has been replicated to the second server.

   As a result, the second server returns the earlier view of the data. *The client application sees data that is different from the data it successfully changed!*

The following sequence diagram illustrates the race condition:



When used in failover mode, also known as active/passive mode, this problem is prevented. However, the load balancer adds network latency while reducing the number of directory servers actively used. This is unfortunate, since the directory server replicas are designed to work together as a pool.

Unlike many load balancers, ForgeRock Identity Platform software has the capability to account for this situation, and to balance the request load appropriately across multiple directory servers.

Recommendations for Load Balancing

Apply the following recommendations in your directory service deployments:

Client is a ForgeRock Identity Platform 6.5 component:

Do not use a load balancer between ForgeRock Identity Platform components and the DS directory service.

ForgeRock Identity Platform components use the same software as DS directory proxy to balance load appropriately across multiple directory servers.

Examples of platform components include AM and IDM.

Client opens a pool of connections to the directory service:

Do not use a load balancer between the client and the DS directory service.

Configure the client application to use multiple directory servers.

Client and server are DS replicas:

Never use a load balancer for replication traffic.

Client can only access a single directory server:

Consider using DS directory proxy server to provide a single point of entry and balance load. Alternatively, use a load balancer in failover or active/passive mode.

Client only ever opens a single connection to the directory service:

Consider using DS directory proxy server to provide a single point of entry and balance load. Alternatively, use a load balancer in failover or active/passive mode.

Appendix B. Getting Support

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.