LDAP SDK Developer's Guide

OpenDJ 2.6

Mark Craig
Ludovic Poitou

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

Abstract

Hands-on guide to developing applications with the OpenDJ SDK. The OpenDJ project offers open source LDAP directory services in Java.

Table of Contents

Preface

This guide shows you how to work with OpenDJ SDK to create client applications in the Java language to connect to LDAP servers and perform LDAP operations.

1. Who Should Read this Guide

This guide is written for Java developers who want to build directory client applications with OpenDJ LDAP SDK.

This guide starts by explaining LDAP directories briefly, and describing best practices for LDAP client applications. Then it demonstrates how to install and use OpenDJ LDAP SDK to build LDAP clients.

You do not need to be an LDAP wizard to learn something from this guide. You do need some background in writing Java 6 and client-server applications to get the most out of this guide. You can nevertheless get started with this guide, and then learn more as you go along.

2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

3. Accessing Documentation Online

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.

4. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

Chapter 1
Understanding LDAP

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, telephone numbers. This means you can look up a directory entry by the name of the user the entry belongs to, but also by her user identifier, her email address, or her telephone number, for example.

1.1. How Directories & 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 brought forth 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 gets pushed out 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 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 OpenDJ code base began in the mid 2000s, when engineers solving the update performance issue decided the cost of adapting the existing C-based directory technology for high performance updates would be higher than the cost of building a next generation, high performance directory using Java technology.

1.2. 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.

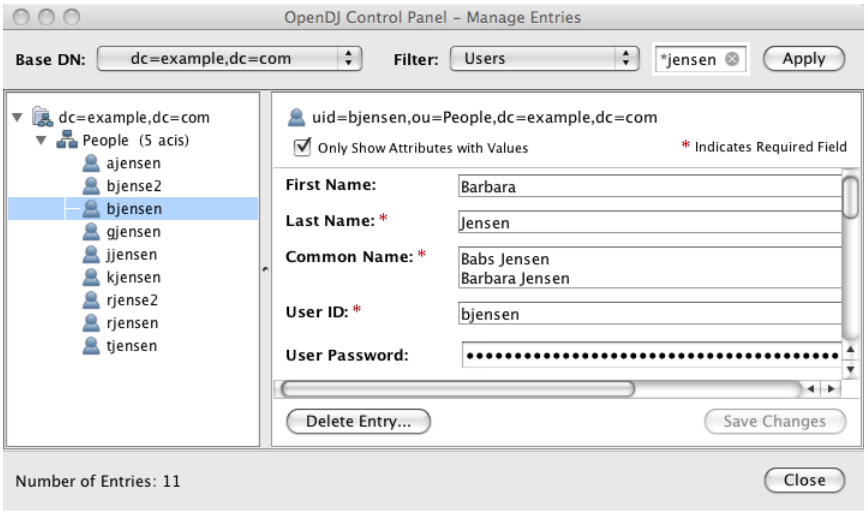
```
dn: uid=bjensen,ou=People,dc=example,dc=com
uid: bjensen
cn: Babs Jensen
cn: Barbara Jensen
facsimileTelephoneNumber: +1 408 555 1992
gidNumber: 1000
givenName: Barbara
homeDirectory: /home/bjensen
l: Cupertino
mail: bjensen@example.com
objectClass: inetOrgPerson
objectClass: organizationalPerson
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`[1]. When you look up her entry in the directory, you specify one or more attributes and values to match in the entries that come back as the result of your search. Typically the attributes you search for are indexed in the directory, so the directory server can retrieve them more quickly.[2]

The entry also has a unique identifier, shown at the top of the entry, `dn: uid=bjensen,ou=People,dc=example,dc=com`. DN stands for distinguished name. No two entries in the directory have the same distinguished name.[3]

[1]The `objectClass` attribute type indicates which types of attributes are allowed and optional 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.
[2]Attribute values do not have to be strings. The directory can use base64 encoding, however, to make binary attribute values, such as passwords, certificates, or photos, portable in text format.
[3]Sometimes your distinguished names include characters that you must escape. The following example shows an entry that includes escaped characters in the DN.

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, looking similar to a pyramid.[4] The distinguished name consists of components separated by commas, `uid=bjensen,ou=People,dc=example,dc=com`. Those components reflect the hierarchy of directory entries.



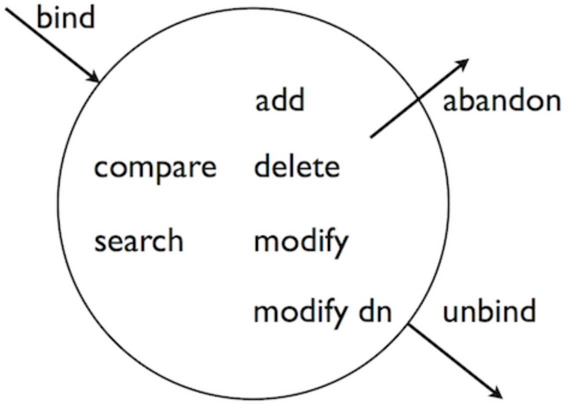
Barbara Jensen's entry is located under an entry with DN `ou=People,dc=example,dc=com`, an organization unit and parent entry for the people at Example.com. The `ou=People` entry is located under the entry with DN `dc=example,dc=com`, the root entry for Example.com. DC stands for domain component. The directory has other root entries, such as `cn=config`, under which the configuration is accessible through LDAP, and potentially others such as `dc=mycompany,dc=com` or `o=myOrganization`. Thus when you look up entries, you specify the parent entry 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.[5]

1.3. LDAP Client & Server Communication

```
$ 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: " # + , ; < = > \
uid: escape
cn: " # + , ; < = > \ DN Escape Characters
sn: DN Escape Characters
mail: escape@example.com
```

[4]Hence pyramid icons are associated with directory servers.
[5]The root entry for the directory, technically the entry with DN `""` (the empty string), is called the root DSE, and contains information about what the server supports, including the other root entries it serves.

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



The standard operations are as follows.

- Bind (authenticate). The first operation in an LDAP session 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.
- 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 lookup all entries for people with email address `bjensen@example.com` in data for Example.com, you would specify a base DN such as `ou=People,dc=example,dc=com` and the filter `(mail=bjensen@example.com)`.
- Compare. After binding, the client can request that the server compare an attribute value the client specifies with the value stored on an entry in the directory.
- Modify. After binding, the client can request that the server change one or more attribute values stored on one or more entries. Often administrators do not allow clients to change directory data, so request that your administrator set appropriate access rights for your client application if you want to update data.
- Add. After binding, the client can request to add one or more new LDAP entries to the server.
- Delete. After binding, the client can request that the server delete one or more entries. To delete and 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. 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 much change distinguished names. [6]

- Unbind. When done making requests, the client should request an unbind operation to release resources right away for other clients.
- 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. The server then drops the connection without a reply to the client.

1.4. Standard LDAPv3 & Extensions

LDAP has standardized two mechanisms for extending the kinds of operations that directory servers can perform. 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.

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 start Transport Layer Security negotiations to protect communications.

Both LDAP controls and extended operations are demonstrated later in this guide. OpenDJ directory server supports many LDAP controls and a few LDAP extended operations, controls and extended operations matching those demonstrated in this guide.

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

Chapter 2
Best Practices For LDAP Application Developers

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

2.1. Authenticate Correctly

Unless your application performs only read operations, you should authenticate to the directory server. Some directory administrators require authentication even to read directory data.

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

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

Your application can use simple, password-based authentication. When you opt for password-based authentication, also use Start TLS for example to avoid sending the password as clear text over the network. If you prefer to manage certificates rather than passwords, directory servers like OpenDJ can do client authentication as well.

2.2. Reuse Connections

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

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

2.3. Health Check Connections

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

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

OpenDJ LDAP SDK offers `Connections.newHeartBeatConnectionFactory()` methods to ensure your `ConnectionFactory` serves connections that are periodically checked to detect whether they are still alive.

2.4. Request Exactly What You Need All At Once

By the time your application makes it to production, you should know what attributes you want, so request them explicitly and request all the attributes you need in the same search. For example, if all you need is `mail` and `cn`, then specify both attributes in your `SearchRequest`.

2.5. Use Specific LDAP Filters

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

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

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

2.6. Make Modifications Specific

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

2.7. Trust Result Codes

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

The LDAP replication model is loosely convergent. In other words, the directory server can, and probably does, send you `ResultCode.SUCCESS` before replicating your change to every directory server instance across the network. If you issue a read immediately after a write, and a load balancer sends your request to another directory server instance, you could get a result that differs from what you expect.

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

2.8. Check Group Membership on the Account, Not the Group

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

2.9. Ask the Directory Server What It Supports

Directory servers expose their capabilities, suffixes they support, and so forth as attribute values on the root DSE. See the section on *Reading Root DSEs*.

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

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

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

2.10. Store Large Attribute Values By Reference

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

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

2.11. Take Care With Persistent Search & Server-Side Sorting

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

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

2.12. Reuse Schemas Where Possible

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

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

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

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

2.13. Handle Referrals

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

2.14. Troubleshooting: Check Result Codes

LDAP result codes are standard and clearly defined. When you receive a `Result`, check the `ResultCode` value to determine what action your application should take. When the result is not what you expect, you can also read or at least log the message string from `ResultCode.getDiagnosticMessage()`.

2.15. Troubleshooting: Check Server Log Files

If you can read the directory server access log, then you can check what the server did with your application's request. For example, the following OpenDJ access log excerpt shows a successful connection from `cn=My Killer App,ou=Apps,dc=example,dc=com` performing a simple bind after Start TLS, and then a simple search before unbind. The lines are wrapped for readability, whereas in the log each record starts with the time stamp.

```
[20/Apr/2012:13:31:05 +0200] CONNECT conn=5
 from=127.0.0.1:51561 to=127.0.0.1:1389 protocol=LDAP
[20/Apr/2012:13:31:05 +0200] EXTENDED REQ conn=5 op=0 msgID=1 name="StartTLS"
 oid="1.3.6.1.4.1.1466.20037"
[20/Apr/2012:13:31:05 +0200] EXTENDED RES conn=5 op=0 msgID=1 name="StartTLS"
 oid="1.3.6.1.4.1.1466.20037" result=0 etime=0
[20/Apr/2012:13:31:07 +0200] BIND REQ conn=5 op=1 msgID=2 version=3 type=SIMPLE
 dn="cn=My Killer App,ou=Apps,dc=example,dc=com"
[20/Apr/2012:13:31:07 +0200] BIND RES conn=5 op=1 msgID=2 result=0
 authDN="cn=My Killer App,ou=Apps,dc=example,dc=com" etime=1
[20/Apr/2012:13:31:07 +0200] SEARCH REQ conn=5 op=2 msgID=3
 base="dc=example,dc=com" scope=wholeSubtree
 filter="(uid=kvaughan)" attrs="isMemberOf"
[20/Apr/2012:13:31:07 +0200] SEARCH RES conn=5 op=2 msgID=3
 result=0 nentries=1 etime=6
[20/Apr/2012:13:31:07 +0200] UNBIND REQ conn=5 op=3 msgID=4
[20/Apr/2012:13:31:07 +0200] DISCONNECT conn=5 reason="Client Unbind"
```

Notice that each operation type is shown in upper case, and that the server tracks both the connection (`conn=5`), operation (`op=[0-3]`), and message ID (`msgID=[1-4]`) numbers to make it easy to filter records. The `etime` refers to how long the server worked on the request in milliseconds. Result code 0 corresponds to `ResultCode.SUCCESS`, as described in RFC 4511.

2.16. Troubleshooting: Inspect Network Traffic

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

Chapter 3
Getting OpenDJ LDAP SDK

This chapter introduces OpenDJ LDAP SDK, demonstrating how to get the software and to build a first basic directory client application.

3.1. About OpenDJ LDAP SDK

OpenDJ LDAP SDK provides a set of modern, developer-friendly Java APIs as part of the OpenDJ product suite. The product suite includes the client SDK alongside command-line tools and sample code, a 100% pure Java directory server, and more. You can use OpenDJ LDAP SDK to create client applications for use with any server that complies with the *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*, RFC 4510.

OpenDJ LDAP SDK brings you easy-to-use connection management, connection pooling, load balancing, and all the standard LDAP operations to read and write directory entries. OpenDJ LDAP SDK also lets you build applications with capabilities defined in additional draft and experimental RFCs that are supported by modern LDAP servers.

3.2. Preparing an LDAP Server

Install an LDAP server such as OpenDJ directory server that you can use to test the applications you develop. Also, load sample data into your server. The sample data used in this guide are available in LDIF form at .

3.3. Getting the LDAP SDK

You can either install a build or build your own from source.

Before you either download a build of OpenDJ LDAP SDK, or get the source code to build your own SDK, make sure you have a Java Development Kit installed. See the *Release Notes* section on *Java Environment* in the *Release Notes* requirements.

Procedure 3.1. To Include the SDK as a Maven Dependency

Follow these steps:

- Include the ForgeRock repository in your list, and include the SDK as a dependency.

```
<repositories>
  <repository>
    <id>forgerock-staging-repository</id>
    <name>ForgeRock Release Repository</name>
    <url>http://maven.forgerock.org/repo/releases</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>forgerock-snapshots-repository</id>
    <name>ForgeRock Snapshot Repository</name>
    <url>http://maven.forgerock.org/repo/snapshots</url>
    <releases>
      <enabled>false</enabled>
    </releases>
  </repository>
</repositories>

...

<dependencies>
  <dependency>
    <groupId>org.forgerock.opendj</groupId>
    <artifactId>opendj-ldap-sdk</artifactId>
    <version>2.6.0</version>
  </dependency>
</dependencies>
```

Procedure 3.2. To Install the Latest SDK & Tools

1. Download the latest OpenDJ LDAP Client Toolkit from the downloads page.

2. Unzip the bundle, `opendj-ldap-toolkit-2.6.0.zip`, where you want to install the SDK.

```
$ unzip opendj-ldap-toolkit-2.6.0.zip
```

3. Add the tools to your PATH.

```
(UNIX)
$ export PATH=/path/to/opendj-ldap-toolkit-2.6.0/bin:$PATH
```

```
(Windows)
C:\>set PATH=\\path\to\opendj-ldap-toolkit-2.6.0\bat:%PATH%
```

4. Add the OpenDJ LDAP SDK for the APIs, the I18N core library, and Grizzly I/O framework for the transport to your CLASSPATH, typically found under `opendj-ldap-toolkit-2.6.0/lib/`.

```
(UNIX)
$ export CLASSPATH=/path/to/lib/grizzly-framework-2.3.jar:$CLASSPATH
$ export CLASSPATH=/path/to/lib/i18n-core-1.4.0.jar:$CLASSPATH
$ export CLASSPATH=/path/to/lib/opendj-ldap-sdk-2.6.0.jar:$CLASSPATH
```

```
(Windows)
C:\>set CLASSPATH=\\path\to\lib\grizzly-framework-2.3.jar:%CLASSPATH%
C:\>set CLASSPATH=\\path\to\lib\i18n-core-1.4.0.jar:%CLASSPATH%
C:\>set CLASSPATH=\\path\to\lib\opendj-ldap-sdk-2.6.0.jar:%CLASSPATH%
```

Procedure 3.3. To Build Your Own SDK From Source

1. Make sure you have Subversion (**svn**) and Maven (**mvn**) installed.
2. Check out the source code.

```
$ svn co https://svn.forgerock.org/opendj/trunk/opendj3
...
Checked out revision XXXX.
```

3. Build the modules and install them in the local repository.

```
$ cd opendj3/
$ mvn install
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] OpenDJ Directory Services Project
[INFO] OpenDJ LDAP SDK
[INFO] OpenDJ LDAP Toolkit
[INFO] OpenDJ LDAP SDK Examples
[INFO] OpenDJ Commons REST Adapter
[INFO] OpenDJ Commons REST LDAP Gateway
[INFO] OpenDJ Server 2.x Adapter
[INFO]
       ...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2:51.315s
[INFO] Finished at: Wed Apr 10 14:28:36 CEST 2013
[INFO] Final Memory: 37M/382M
[INFO] ------------------------------------------------------------------------
```

4. Unzip the tools and libraries included in the file, `opendj3/opendj-ldap-toolkit/target/opendj-ldap-toolkit-2.6.0.zip`.
5. Add the `opendj-ldap-toolkit-2.6.0/bin` (UNIX) or `opendj-ldap-toolkit-2.6.0\bat` (Windows) directory to your PATH.
6. Set your CLASSPATH to include the OpenDJ LDAP SDK library, `opendj-ldap-sdk-2.6.0.jar`, the I18N core library, `i18n-core-1.4.0.jar`, and the Grizzly framework, `grizzly-framework-2.3.jar` under `opendj-ldap-toolkit-2.6.0/lib/`.

After you install OpenDJ LDAP SDK and configure your environment as described, if you have a directory server running import sample data, and test your configuration with a sample client application.

```
import org.forgerock.opendj.ldap.Connection;
import org.forgerock.opendj.ldap.LDAPConnectionFactory;
import org.forgerock.opendj.ldap.SearchScope;
import org.forgerock.opendj.ldap.responses.SearchResultEntry;
import org.forgerock.opendj.ldap.responses.SearchResultReference;
import org.forgerock.opendj.ldif.ConnectionEntryReader;
import org.forgerock.opendj.ldif.LDIFEntryWriter;

//Test.java:
//Kick the SDK tires, reading Babs Jensen's entry and displaying LDIF.
//If your LDAP server is not listening on localhost:1389, or if your
//data are different change the appropriate lines below.

class Test {
    public static void main(String[] args) {

        // Create an LDIF writer which will write the search results to stdout.

        final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
        Connection connection = null;

        try {
            // Connect and bind to the server.
            // CHANGE THIS IF SERVER IS NOT AT localhost:1389.
            final LDAPConnectionFactory factory =
                    new LDAPConnectionFactory("localhost", 1389);

            connection = factory.getConnection();
            // CHANGE THIS IF ANONYMOUS SEARCHES ARE NOT ALLOWED.
            // connection.bind(userName, password);

            // Read the entries and output them as LDIF.
            // CHANGE THIS IF NO uid=bjensen,ou=people,dc=example,dc=com EXISTS.
            final ConnectionEntryReader reader =
                    connection.search("dc=example,dc=com",
                            SearchScope.WHOLE_SUBTREE, "(uid=bjensen)", "*");
            while (reader.hasNext()) {
                if (reader.isEntry()) {
                    // Got an entry.
                    final SearchResultEntry entry = reader.readEntry();
                    writer.writeComment("Search result entry: "
                            + entry.getName().toString());
                    writer.writeEntry(entry);
                } else {
                    // Got a continuation reference.
                    final SearchResultReference ref = reader.readReference();
                    writer.writeComment("Search result reference: "
                            + ref.getURIs().toString());
                }
            }
            writer.flush();
        } catch (final Exception e) {
            // Handle exceptions...
            System.err.println(e.getMessage());
        } finally {
            if (connection != null) {
                connection.close();
            }
        }
    }
}
```

If all goes well, `Test.java` compiles without errors. The test program displays Babs Jensen's entry in LDIF.

```
$ javac Test.java
$ java Test
# Search result entry: uid=bjensen,ou=People,dc=example,dc=com
dn: uid=bjensen,ou=People,dc=example,dc=com
givenName: Barbara
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
uid: bjensen
cn: Barbara Jensen
cn: Babs Jensen
sn: Jensen
telephoneNumber: +1 408 555 1862
roomNumber: 0209
ou: Product Development
ou: People
l: Cupertino
mail: bjensen@example.com
facsimileTelephoneNumber: +1 408 555 1992
```

Chapter 4 Using the LDAP SDK

As LDAP relies on a connection from the client to the directory server, the starting point for working with the LDAP SDK is a new `LDAPConnectionFactory`, from which you then get either a synchronous connection, or pass in a handler to an asynchronous connection. You then use the connection to make requests and get responses from the directory server.

4.1. Synchronous & Asynchronous Operations

For synchronous operations your application gets a connection from the `LDAPConnectionFactory` and requests operations on the connection. When finished, your application closes the connection.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;

try {
    connection = factory.getConnection();

    // Perform operations on the connection, such as connection.bind(),
    // connection.search(), connection.modify(), etc.

    } catch (final ErrorResultException e) {
        System.err.println(e.getMessage());
        System.exit(e.getResult().getResultCode().intValue());
        return;
    } finally {
        if (connection != null) {
            connection.close();
        }
    }
```

For asynchronous operations, your application passes a result handler to `LDAPConnectionFactory.getConnectionAsync()` that implements the `ResultHandler<Connection>` interface.

```
private static final class ConnectResultHandlerImpl
        implements ResultHandler<Connection> {
    @Override
    public void handleErrorResult(final ErrorResultException error) {
        ...
    }

    @Override
    public void handleResult(final Connection connection) {
        // Connect succeeded: save connection and initiate bind.
        SearchAsync.connection = connection;

        final BindRequest request =
                Requests.newSimpleBindRequest(userName, password.toCharArray());
        connection.bindAsync(request, null, new BindResultHandlerImpl());
    }
}

// Main method initiates async operations by getting a connection...
final LDAPConnectionFactory factory = new LDAPConnectionFactory(hostName, port);
factory.getConnectionAsync(new ConnectResultHandlerImpl());

...

if (connection != null) {
    connection.close();
}
```

When the connection result handler gets a connection, your application can pass result handlers for other operations using methods on the connection named `*Async()`. For most operations, your application implements `ResultHandler`. For searches, your application implements `SearchResultHandler`. The result handler is notified upon completion of the operation.

Asynchronous methods are non-blocking, returning a `FutureResult` whose `get()` method lets you retrieve the result. Your application must coordinate concurrency when you use asynchronous operations.

4.2. Managing Errors

LDAP defines many result codes to deal with conditions other than success. The `ResultCode` class encapsulates the LDAP codes and additional client-side codes specific to the SDK.

Your application deals with most non-success result codes when it catches one of the LDAP SDK exceptions corresponding to the operation you requested. `ErrorResultException` is a common way for the SDK to indicate a non-successful result. Your application can then take remedial action based on the result code, as in the following synchronous excerpt.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;

try {
    connection = factory.getConnection();
    connection.bind(name, password);

    // Perform operations on the connection...

} catch (final ErrorResultException e) {

    // Take remedial action based on the result code...
    // e.getResult().getResultCode() returns the code for you to interpret.

} finally {
    if (connection != null) {
        connection.close();
    }
}
```

Also notice the methods `ResultCode.getName()` that provides a short, human-readable version of the result code, and `Result.getDiagnosticMessage()` that can also help debug problems after the fact.

Chapter 5

Authenticating To the Directory

When your client application connects to the directory, the first operation to perform is a bind operation. The bind operation authenticates the client to the directory.

5.1. Simple Authentication

You perform simple authentication by binding with the distinguished name of a user's directory entry and the user's password. For this reason simple authentication over unsecure network connections should be done only in the lab. If your real end users are providing their passwords, your application must use simple authentication only if the network is secure.

To bind using Barbara Jensen's identity and simple authentication, for example, your application would provide the DN `uid=bjensen,ou=People,dc=example,dc=com` with the password `hifalutin`.

The directory stores the password value used for simple authentication in binary form on the `userPassword` attribute of the entry. In other words, for the purposes of your application the password is not a string, but instead an array of bytes. Typically the directory is further configured to store only hashed values of user passwords, rather than plain text versions. Thus even if someone managed to read the stored password values, they would still have to crack the hash in order to learn the actual passwords. When your application performing simple authentication sends the password value, the directory server therefore hashes the password value, and then compares the hashed result with the value of the `userPassword` on the user entry. If the values match, then the directory authenticates the user. Once the user has authenticated, the directory determines authorization for operations on the connection based on the users identity.

```
/**
 * Authenticate over LDAP.
 */
private static void connect()
{
  final LDAPConnectionFactory factory = new LDAPConnectionFactory(
    host, port);
  Connection connection = null;

  try
  {
    connection = factory.getConnection();
    connection.bind(bindDN, bindPassword.toCharArray());
    System.out.println("Authenticated as " + bindDN + ".");
  }
  catch (final ErrorResultException e)
  {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
  }
  finally
  {
    if (connection != null) connection.close();
  }
}
```

If the password values do not match, a directory might nevertheless authenticate the client application. The LDAP specifications say that in this case, however, the directory authenticates the user as anonymous, therefore no doubt with fewer rights than the normal user, and surely fewer rights than an administrator.

5.2. Start TLS & SSL Authentication

Simple authentication involves sending a user name and password to the directory server. To avoid sending the user name and password in the clear, you can use SSL or Start TLS.

For both SSL and Start TLS, you pass LDAP options to the connection factory in order to set an SSL context, and set whether to use Start TLS. The SSL context lets you set a trust manager to check server certificates, and also set a key manager to provide keys when the server needs to check your client certificates. In the simplest, not-so-secure case, you can set up a trust manager that trusts all certificates.

The following example is an excerpt from the OpenDJ LDAP SDK example, `org.forgerock.opendj.examples.SimpleAuth.java`.

```
private static LDAPOptions getTrustAllOptions()
  throws GeneralSecurityException
{
  LDAPOptions lo = new LDAPOptions();
  SSLContext sslContext = new SSLContextBuilder()
    .setTrustManager(TrustManagers.trustAll()).getSSLContext();
  lo.setSSLContext(sslContext);
  lo.setUseStartTLS(useStartTLS);
  return lo;
}
```

A more secure and extensive SSL context would include a trust manager using a trust store and trust manager methods to check server certificates. If you also want to be able to authenticate to the server using your client certificate, you would need a key manager.

The authentication over SSL or using Start TLS in the trust-all case is much like simple authentication over LDAP without connection-level security. The primary differences are that you pass the `LDAPOptions` to the LDAP connection factory, and that you handle the potential security exception involved in setting up the SSL context.

```
/**
 * Perform authentication over a secure connection, trusting all server
 * certificates.
 */
private static void trustAllConnect()
{
  Connection connection = null;

  try
  {
    final LDAPConnectionFactory factory =
        new LDAPConnectionFactory(host, port, getTrustAllOptions());
    connection = factory.getConnection();
    connection.bind(bindDN, bindPassword.toCharArray());
    System.out.println("Authenticated as " + bindDN + ".");
  }
  catch (final ErrorResultException e)
  {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
  }
  catch (final GeneralSecurityException e)
  {
    System.err.println(e.getMessage());
    System.exit(ResultCode.CLIENT_SIDE_CONNECT_ERROR.intValue());
  }
  finally
  {
    if (connection != null)
      connection.close();
  }
}
```

5.3. SASL Authentication

Simple Authentication and Security Layer (SASL) provides a way to use other mechanisms for authentication such as Kerberos or Digest authentication, or even to define your own authentication mechanism. The directory server likely advertises supported SASL mechanisms in the root DSE. The follow example shows how to search OpenDJ for supported SASL mechanisms.

```
$ ldapsearch
 --port 1389
 --bindDN "cn=Directory Manager"
 --bindPassword password
 --baseDN ""
 --searchScope base
 "(objectclass=*)" supportedSASLMechanisms
dn:
supportedSASLMechanisms: PLAIN
supportedSASLMechanisms: EXTERNAL
supportedSASLMechanisms: DIGEST-MD5
supportedSASLMechanisms: CRAM-MD5
```

Notice that neither the Kerberos (GSSAPI SASL) nor the Anonymous mechanism is enabled by default, though OpenDJ implements both.

In order to use a SASL mechanism to bind, your program must set up a `SASLBindRequest` and pass that to the `bind()` method of the `Connection`.

This section shows an example using the SASL PLAIN mechanism, which takes either a DN or a user ID to authenticate, with an optional DN or user ID as the authorization ID that identifies the user who performs operations. The SASL PLAIN mechanism itself does not secure the connection, so the example uses StartTLS. The example is provided with the OpenDJ LDAP SDK examples in `org.forgerock.opendj.examples.SASLAuth.java`. The following excerpt shows the core of the bind process.

```

try
{
  final LDAPConnectionFactory factory =
      new LDAPConnectionFactory(host, port, getTrustAllOptions());
  connection = factory.getConnection();
  PlainSASLBindRequest request =
      Requests.newPlainSASLBindRequest(authcid, passwd.toCharArray())
      .setAuthorizationID(authzid);
  connection.bind(request);
  System.out.println("Authenticated as " + authcid + ".");
}
```

The implementation for `getTrustAllOptions()`, the same as in the example above, sets up Start TLS. When you run this example with both authorization and authentication IDs, `authzid` and `authcid`, set to `u:bjensen` and password `hifalutin`, the bind is successful, and the program reaches the final line of the `try` block.

```
Authenticated as u:bjensen.
```

Behind the scenes, OpenDJ has the SASL PLAIN mechanism configured by default to use the Exact Match Identity Mapper to look up user IDs as `uid` values. If you use another directory server, you might have to configure how it maps user IDs to user entries.

Chapter 6
Searching & Comparing Directory Data

Traditionally directories excel at serving read requests. This chapter covers the read (search and compare) capabilities that OpenDJ LDAP Java SDK provides. The data used in examples here is .

6.1. About Searching

An LDAP search looks up entries based on the following parameters.

- A *filter* that indicates which attribute values to match
- A *base DN* that specifies where in the directory information tree to look for matches
- A *scope* that defines how far to go under the base DN
- A list of attributes to fetch for an entry when a match is found

For example, imagine you must write an application where users login using their email address and a password. After the user logs in, your application displays the user's full name so it is obvious who is logged in. Your application is supposed to go to the user directory both for authentication, and also to read user profile information. You are told the user directory stores user profile entries under base DN `ou=People,dc=example,dc=com`, that email addresses are stored on the standard `mail` attribute, and full names are store on the standard `cn` attribute.

You figure out how to authenticate from the chapter on authentication, in which you learn you need a bind DN and a password to do simple authentication. But how do you find the bind DN given the email? How do you get the full name?

The answer to both questions is that you do an LDAP search for the user's entry, which has the DN that you use to bind, and you have the server fetch the `cn` attribute in the results. Your search uses the following parameters.

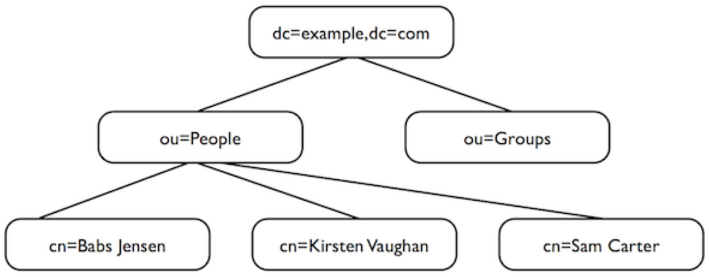
- The filter is `(mail=`*`emailAddress`*`)`, where *emailAddress* is the email address the user provided.
- The base DN is the one given to you, `ou=People,dc=example,dc=com`.
- For the scope, you figure the user entry is somewhere under the base DN, so you opt to search the whole subtree.
- The attribute to fetch is `cn`.

The following code excerpt demonstrates how this might be done in a minimal command-line program.

```
// Prompt for mail and password.
Console c = System.console();
if (c == null) {
    System.err.println("No console.");
    System.exit(1);
}

String mail = c.readLine("Email address: ");
char[] password = c.readPassword("Password: ");

// Search using mail address, and then bind with the DN and password.
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host,
        port);
Connection connection = null;
try {
    connection = factory.getConnection();

    // No explicit bind yet so we remain anonymous for now.
    SearchResultEntry entry = connection.searchSingleEntry(baseDN,
            SearchScope.WHOLE_SUBTREE, "(mail=" + mail + ")", "cn");
    DN bindDN = entry.getName();
    connection.bind(bindDN.toString(), password);

    String cn = entry.getAttribute("cn").firstValueAsString();
    System.out.println("Hello, " + cn + "!");
} catch (final ErrorResultException e) {
    System.err.println("Failed to bind.");
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

6.2. Setting Search Base & Scope

Directory servers organize entries somewhat like a file system. Directory data is often depicted as an upside-down tree.



In the figure shown above, entries are represented by the relevant parts of their DNs. The entry with DN `dc=example,dc=com` is the base entry for a suffix. Under the base entry, you see two organizational units, one for people, `ou=People`, the other for groups, `ou=Groups`. The entries for people include those of Babs Jensen, Kirsten Vaughan, and Sam Carter.

When you are searching for a person's entry somewhere under `dc=example,dc=com`, you can start from `dc=example,dc=com`, from `ou=People,dc=example,dc=com`, or if you have enough information to pinpoint the user entry and only want to look up another attribute value for example, then directly from the entry such as `cn=Babs Jensen,ou=People,dc=example,dc=com`. The DN of the entry where you choose to start the search is the base DN for the search.

When searching, you also define the scope. Scope defines what entries the server considers when checking for entries that match your search.

- For `SearchScope.BASE_OBJECT` the server considers only the base entry.

  This is the scope you use if you know the full DN of the object that interests you. For example, if your base DN points to Babs Jensen's entry, `cn=Babs Jensen,ou=People,dc=example,dc=com`, and you want to read some of Babs's attributes, you would set scope to `SearchScope.BASE_OBJECT`.

- For `SearchScope.SINGLE_LEVEL` the server considers all entries directly below the base entry.

  You use this scope if for example you want to discover organizational units under `dc=example,dc=com`, or if you want to find people's entries and you know they are immediately under `ou=People,dc=example,dc=com`.

- For `SearchScope.SUBORDINATES` the server considers all entries below the base entry.

  This scope can be useful if you know that the base DN for your search is an entry that you do not want to match.

- For `SearchScope.WHOLE_SUBTREE` (default) the server considers the base entry and all entries below.

In addition to a base DN and scope, a search request also calls for a search filter.

6.3. Working With Search Filters

When you look someone up in the telephone directory, you use the value of one attribute of a person's entry (last name), to recover the person's directory entry, which has other attributes (phone number, address). LDAP works the same way. In LDAP, search requests identify both the scope of the directory entries to consider (for example, all people or all organizations), and also the entries to retrieve based on some attribute value (for example, surname, mail address, phone number, or something else). The way you express the attribute value(s) to match is by using a search filter.

LDAP search filters define what entries actually match your request. For example, the following simple equality filter says, "Match all entries that have a surname attribute (sn) value equivalent to Jensen."

```
(sn=Jensen)
```

When you pass the directory server this filter as part of your search request, the directory server checks the entries in scope for your search to see whether they match.[1] If the directory server finds entries that match, it returns those entries as it finds them.

The example, `(sn=Jensen)`, shows a string representation of the search filter. The OpenDJ LDAP SDK lets you express your filters as strings, or as `Filter` objects. In both cases, the SDK translates the strings and objects into the binary representation sent to the server over the network.

Equality is just one of the types of comparisons available in LDAP filters. Comparison operators include the following.

Table 6.1. LDAP Filter Operators

Operator	Definition	Example
`=`	Equality comparison, as in `(sn=Jensen)`. This can also be used with substring matches. For example, to match last names starting with `Jen`, use the filter `(sn=Jen*)`. Substrings are more expensive for the directory server to index. Substring searches therefore might not be permitted for many attributes.	`"(cn=My App)"` matches entries with common name `My App`. `"(sn=Jen*)"` matches entries with surname starting with `Jen`.
`<=`	Less than or equal to comparison, which works alphanumerically.	`"(cn<=App)"` matches entries with `commonName` up to those starting with App (case-insensitive) in alphabetical order.
`>=`	Greater than or equal to comparison, which works alphanumerically.	`"(uidNumber>=1151)"` matches entries with `uidNumber` greater than 1151.
`=*`	Presence comparison. For example, to match all entries having a `userPassword`, use the filter `(userPassword=*)`.	`"(member=*)"` matches entries with a `member` attribute.
`~=`	Approximate comparison, matching attribute values similar to the value you specify.	`"(sn~=jansen)"` matches entries with a surname that sounds similar to `Jansen` (Johnson, Jensen, and so forth).
`[:dn][:`*`oid`*`]:=`	Extensible match comparison. At the end of the OID or language subtype, you further specify the matching rule as follows: • Add `.1` for less than • Add `.2` for less than or equal to • Add `.3` for equal to (default) • Add `.4` for greater than or equal to	`(uid:dn:=bjensen)` matches entries where `uid` having the value `bjensen` is a component of the entry DN. `(lastLoginTime: 1.3.6.1.4.1.26027.1.4.5:= -13w)` matches entries with a last login time more recent than 13 weeks. You also use extensible match filters with localized values. Directory servers like OpenDJ support a variety of internationalized locales, each of which has an OID for collation

[1]In fact, the directory server probably checks an index first, and might not even accept search requests unless it can use indexes to match your filter rather than checking all entries in scope.

Operator	Definition	Example
	• Add `.5` for greater than • Add `.6` for substring	order, such as `1.3.6.1.4.1.42.2.27.9.4.76.1` for French. OpenDJ also lets you use the language subtype, such as `fr`, instead of the OID. `"(cn:dn:=My App)"` matches entries who have `My App` as the common name and also as the value of a DN component.
`!`	NOT operator, to find entries that do not match the specified filter component. Take care to limit your search when using `!` to avoid matching so many entries that the server treats your search as unindexed.	`'!(objectclass=person)'` matches non-person entries.
`&`	AND operator, to find entries that match all specified filter components.	`'(&(l=Cupertino)(!(uid=bjensen)))'` matches entries for users in Cupertino other than the user with ID `bjensen`.
`|`	OR operator, to find entries that match one of the specified filter components.	`"|(sn=Jensen)(sn=Johnson)"` matches entries with surname Jensen or surname Johnson.

When taking user input, take care to protect against users providing input that has unintended consequences. OpenDJ SDK offers several Filter methods to help you. First, you can use strongly typed construction methods such as `Filter.equality()`.

```
String userInput = getUserInput();
Filter filter = Filter.equality("cn", userInput);

// Invoking filter.toString() with input of "*" results in a filter
// string "(cn=\2A)".
```

You can also let the SDK escape user input by using a template with `Filter.format()` as in the following example.

```
String template = "(|(cn=%s)(uid=user.%s))";
String[] userInput = getUserInput();
Filter filter = Filter.format(template, userInput[0], userInput[1]);
```

Finally, you can explicitly escape user input with `Filter.escapeAssertionValue()`.

```
String baseDN = "ou=people,dc=example,dc=com";
String userInput = getUserInput();

// Filter.escapeAssertionValue() transforms user input of "*" to "\2A".
SearchRequest request = Requests.newSearchRequest(
        baseDN, SearchScope.WHOLE_SUBTREE,
        "(cn=" + Filter.escapeAssertionValue(userInput) + "*)", "cn", "mail");
```

6.4. Sending a Search Request

As shown in the following excerpt with a synchronous connection, you get a `Connection` to the directory server from an `LDAPConnectionFactory`.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host,
        port);
Connection connection = null;
try {
    connection = factory.getConnection();

    // Do something with the connection...
} catch (Exception e) {
    // Handle exceptions...
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

The `Connection` gives you `search()` methods that either take parameters in the style of the **ldapsearch** command, or that take a `SearchRequest` object. If you are sure that the search only returns a single entry, you can read the entry with the `searchSingleEntry()` methods. If you have the distinguished name, you can use `readEntry()` directly.

6.5. Getting Search Results

Depending on the method you use to search, you handle results in different ways.

- You can get a `ConnectionEntryReader`, and iterate over the reader to access individual search results.

```
Connection connection = ...;
ConnectionEntryReader reader = connection.search("dc=example,dc=com",
    SearchScope.WHOLE_SUBTREE, "(objectClass=person)");
try
{
  while (reader.hasNext())
  {
    if (reader.isEntry())
    {
      SearchResultEntry entry = reader.readEntry();

      // Handle entry...
    }
    else
    {
      SearchResultReference ref = reader.readReference();

      // Handle continuation reference...
    }
  }
}
catch (IOException e)
{
  // Handle exceptions...
}
finally
{
  reader.close();
}
```

- You can pass in a collection of `SearchResultEntry`s (and optionally a collection of `SearchResultReference`s) to which the SDK adds the results. For this to work, you need enough memory to hold everything the search returns.
- You can pass in a `SearchResultHandler` to manage results.
- With `searchSingleEntry()` and `readEntry()`, you can get a single `SearchResultEntry` with methods to access the entry content.

6.6. Working With Entry Attributes

When you get an entry object, chances are you want to handle attribute values as objects. The OpenDJ LDAP SDK provides the `Entry.parseAttribute()` method and an `AttributeParser` with methods for a variety of attribute value types. You can use these methods to get attribute values as objects.

```

// Use Kirsten Vaughan's credentials and her entry.
String name = "uid=kvaughan,ou=People,dc=example,dc=com";
char[] password = "bribery".toCharArray();
connection.bind(name, password);

// Make sure we have a timestamp to play with.
updateEntry(connection, name, "description");
```

```
// Read Kirsten's entry.
final SearchResultEntry entry = connection.readEntry(name,
        "cn", "objectClass", "hasSubordinates", "numSubordinates",
        "isMemberOf", "modifyTimestamp");

// Get the entry DN and some attribute values as objects.
DN dn = entry.getName();

Set<String> cn = entry.parseAttribute("cn").asSetOfString("");
Set<AttributeDescription> objectClasses =
        entry.parseAttribute("objectClass").asSetOfAttributeDescription();
boolean hasChildren = entry.parseAttribute("hasSubordinates").asBoolean();
int numChildren = entry.parseAttribute("numSubordinates").asInteger(0);
Set<DN> groups = entry
        .parseAttribute("isMemberOf")
        .usingSchema(Schema.getDefaultSchema()).asSetOfDN();
Calendar timestamp = entry
        .parseAttribute("modifyTimestamp")
        .asGeneralizedTime().toCalendar();

// Do something with the objects.
// ...
```

6.7. Working With LDAP URLs

LDAP URLs express search requests in URL form. In the directory data you can find them used as `memberURL` attribute values for dynamic groups, for example. The following URL from the configuration for the administrative backend lets the directory server build a dynamic group of administrator entries that are children of `cn=Administrators,cn=admin data`.

```
ldap:///cn=Administrators,cn=admin data??one?(objectclass=*)
```

The static method `LDAPUrl.valueOf()` takes an LDAP URL string and returns an `LDAPUrl` object. You can then use the `LDAPUrl.asSearchRequest()` method to get the `SearchRequest` that you pass to one of the search methods for the connection.

6.8. Sorting Search Results

If you want to sort search results in your client application, then make sure you have enough memory in the JVM to hold the results of the search, and use one of the search methods that lets you pass in a collection of `SearchResultEntry`s. After the collection is populated with the results, you can sort them.

If you are on good terms with your directory administrator, you can perhaps use a server-side sort control. The server-side sort request control asks the server to sort the results before returning them, and so is a memory intensive operation on the directory server. You set up the control using `ServerSideSortRequestControl.newControl()`. You get the control into your search by building a search request to pass to the search method, using `SearchRequest.addControl()` to attach the control before passing in the request.

If your application needs to scroll through search results a page at a time, work with your directory administrator to set up the virtual list view indexes that facilitate scrolling through results.

6.9. About Comparing

You use the LDAP compare operation to make an assertion about an attribute value on an entry. Unlike the search operation, you must know the distinguished name of the entry in advance to request a compare operation. You also specify the attribute type name and the value to compare to the values stored on the entry.

`Connection` has a choice of compare methods, depending on how you set up the operation.

Check the `ResultCode` from `CompareResult.getResultCode()` for `ResultCode.COMPARE_TRUE` or `ResultCode.COMPARE_FALSE`.

Chapter 7 Getting Information About the Directory Service

LDAP directories expose what their capabilities through the root DSE. They also expose their schema definitions, which define the sort of entries and attributes can be stored in a directory, over protocol. OpenDJ SDK allows you to look up that information in your client application.

7.1. Reading Root DSEs

The directory entry with distinguished name `""` (empty string) is called the *root DSE*. DSE stands for DSA-Specific Entry. DSA stands for Directory Server Agent, a single directory server.

The root DSE serves to expose information over LDAP about what the directory server supports in terms of LDAP controls, auth password schemes, SASL mechanisms, LDAP protocol versions, naming contexts, features, LDAP extended operations, and so forth. The root DSE holds all the information as values of LDAP attributes. OpenDJ defines these attributes as operational. In other words, OpenDJ only returns the attributes if you either request them specifically, or request all operational attributes.

To access the list of what an OpenDJ server supports, for example, get all operational attributes from the root DSE entry as in the following excerpt.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(
  host, port);
Connection connection = null;

try
{
  connection = factory.getConnection();

  // Perform an anonymous search on the root DSE.
  final SearchResultEntry entry = connection.searchSingleEntry(
      "",                          // DN is "" for root DSE.
      SearchScope.BASE_OBJECT,     // Read only the root DSE.
      "objectclass=*",             // Every object matches this filter.
      "+");                        // Return all operational attributes.

  final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
  writer.writeComment("Root DSE for LDAP server at " + host + ":" + port);
  if (entry != null) writer.writeEntry(entry);
  writer.flush();
}
```

Notice that by default you can access the root DSE after authenticating anonymously. When you look at the entry in LDIF, you see that supported capabilities are generally identified by object identifier (OID).

```
# Root DSE for LDAP server at localhost:1389
dn:
supportedControl: 1.2.826.0.1.3344810.2.3
supportedControl: 1.2.840.113556.1.4.1413
supportedControl: 1.2.840.113556.1.4.319
supportedControl: 1.2.840.113556.1.4.473
supportedControl: 1.2.840.113556.1.4.805
supportedControl: 1.3.6.1.1.12
supportedControl: 1.3.6.1.1.13.1
supportedControl: 1.3.6.1.1.13.2
supportedControl: 1.3.6.1.4.1.26027.1.5.2
supportedControl: 1.3.6.1.4.1.42.2.27.8.5.1
supportedControl: 1.3.6.1.4.1.42.2.27.9.5.2
supportedControl: 1.3.6.1.4.1.42.2.27.9.5.8
supportedControl: 1.3.6.1.4.1.4203.1.10.1
supportedControl: 1.3.6.1.4.1.4203.1.10.2
supportedControl: 1.3.6.1.4.1.7628.5.101.1
supportedControl: 2.16.840.1.113730.3.4.12
supportedControl: 2.16.840.1.113730.3.4.16
supportedControl: 2.16.840.1.113730.3.4.17
supportedControl: 2.16.840.1.113730.3.4.18
supportedControl: 2.16.840.1.113730.3.4.19
supportedControl: 2.16.840.1.113730.3.4.2
supportedControl: 2.16.840.1.113730.3.4.3
supportedControl: 2.16.840.1.113730.3.4.4
supportedControl: 2.16.840.1.113730.3.4.5
supportedControl: 2.16.840.1.113730.3.4.9
supportedAuthPasswordSchemes: MD5
supportedAuthPasswordSchemes: SHA1
supportedAuthPasswordSchemes: SHA256
supportedAuthPasswordSchemes: SHA512
supportedAuthPasswordSchemes: SHA384
supportedSASLMechanisms: PLAIN
supportedSASLMechanisms: EXTERNAL
supportedSASLMechanisms: DIGEST-MD5
supportedSASLMechanisms: CRAM-MD5
supportedLDAPVersion: 2
supportedLDAPVersion: 3
etag: 00000000e9155ba0
pwdPolicySubentry: cn=Default Password Policy,cn=Password Policies,cn=config
supportedFeatures: 1.3.6.1.1.14
supportedFeatures: 1.3.6.1.4.1.4203.1.5.1
supportedFeatures: 1.3.6.1.4.1.4203.1.5.2
supportedFeatures: 1.3.6.1.4.1.4203.1.5.3
subschemaSubentry: cn=schema
changelog: cn=changelog
supportedTLSCiphers: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_RSA_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
supportedTLSCiphers: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
supportedTLSCiphers: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
```

```
supportedTLSCiphers: TLS_RSA_WITH_AES_128_CBC_SHA
supportedTLSCiphers: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
supportedTLSCiphers: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
supportedTLSCiphers: TLS_DHE_RSA_WITH_AES_128_CBC_SHA
supportedTLSCiphers: TLS_DHE_DSS_WITH_AES_128_CBC_SHA
supportedTLSCiphers: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
supportedTLSCiphers: TLS_ECDHE_RSA_WITH_RC4_128_SHA
supportedTLSCiphers: SSL_RSA_WITH_RC4_128_SHA
supportedTLSCiphers: TLS_ECDH_ECDSA_WITH_RC4_128_SHA
supportedTLSCiphers: TLS_ECDH_RSA_WITH_RC4_128_SHA
supportedTLSCiphers: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: SSL_RSA_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
supportedTLSCiphers: SSL_RSA_WITH_RC4_128_MD5
supportedTLSCiphers: TLS_EMPTY_RENEGOTIATION_INFO_SCSV
ds-private-naming-contexts: cn=admin data
ds-private-naming-contexts: cn=ads-truststore
ds-private-naming-contexts: cn=backups
ds-private-naming-contexts: cn=config
ds-private-naming-contexts: cn=monitor
ds-private-naming-contexts: cn=schema
ds-private-naming-contexts: cn=tasks
ds-private-naming-contexts: dc=replicationChanges
supportedTLSProtocols: SSLv2Hello
supportedTLSProtocols: SSLv3
supportedTLSProtocols: TLSv1
supportedTLSProtocols: TLSv1.1
supportedTLSProtocols: TLSv1.2
numSubordinates: 1
namingContexts: dc=example,dc=com
structuralObjectClass: ds-root-dse
lastExternalChangelogCookie:
lastChangeNumber: 0
firstChangeNumber: 0
supportedExtension: 1.3.6.1.1.8
supportedExtension: 1.3.6.1.4.1.26027.1.6.1
supportedExtension: 1.3.6.1.4.1.26027.1.6.2
supportedExtension: 1.3.6.1.4.1.26027.1.6.3
supportedExtension: 1.3.6.1.4.1.4203.1.11.1
supportedExtension: 1.3.6.1.4.1.1466.20037
supportedExtension: 1.3.6.1.4.1.4203.1.11.3
vendorName: ForgeRock AS.
vendorVersion: OpenDJ 2.5.0
hasSubordinates: true
entryDN:
entryUUID: d41d8cd9-8f00-3204-a980-0998ecf8427e
```

Three key pieces of information in the entry shown above are attribute values for `namingContexts` (showing the base DNs under which your application can look for user data), `subschemaSubentry` (indicating where the LDAP schema are stored), and `supportedLDAPVersion` (with OpenDJ seen to support both LDAPv2 and LDAPv3).

7.2. Checking For LDAPv3 Support

As shown in the previous section, you can check that the root DSE attribute `supportedLDAPVersion` has a value of 3.

LDAPv3 has been available since 1997. Client applications built with OpenDJ SDK use LDAPv3.

7.3. Getting Schema Information

The root DSE attribute `subschemaSubentry` shows the DN of the entry holding LDAP schema definitions. LDAP schema defines the object classes, attributes types, attribute value syntaxes, matching rules and so on that constrain entries held by the LDAP server.

The `org.forgerock.opendj.ldap.schema` package is devoted to constructing and querying LDAP schemas. The `Schema` class for example lets you `readSchemaForEntry()` to get the relevant schema from the subschema subentry, and then `validateEntry()` to check an entry your application has constructed before sending the entry to the server.

Chapter 8
Updating Directory Data

Modern directory servers like OpenDJ can handle a high load of write requests, replicating changes quickly both on the LAN and over the WAN.

8.1. About Add, Modify, Rename, & Delete

The four basic CRUD operations — create, read, update, and delete — correspond to the LDAP operations add, search, modify (or modify DN), and delete.[1]

- An add request is used to create a new entry in an LDAP directory. The entry must have a unique distinguished name that belongs under a base DN served by the directory. The entry must have a list of attributes that are valid according to the directory schema.
- Search requests are described in the chapter on *Searching & Comparing Directory Data*.
- A modify request is used to add, delete, or replace attribute values on an entry in an LDAP directory. The resulting entry must be valid according to the directory schema.

  A modify DN request is used to rename or move a directory entry. In both cases the distinguished name changes. Renaming involves changing the relative distinguished name, for example from `cn=Bob,ou=People,dc=example,dc=com` to `cn=Ted,ou=People,dc=example,dc=com`. Moving involves changing the container where the entry is found, for example from `cn=Barbara Jensen,ou=People,dc=Old Company,dc=com` to `cn=Barbara Jensen,ou=People,dc=New Company,dc=com`.

  Although they are both considered modify DN operations, renaming a leaf entry is generally much simpler than moving a container entry that has child entries. Not all modify DN operations mobilize equivalent resources on the directory server.
- A delete request is used to remove an entry from an LDAP directory.

  Directory servers can restrict deletes to leaf entries, so that you cannot remove an entry that has other child entries. For example, you have to delete `uid=bjensen,ou=People,dc=example,dc=com` and other peer entries before you delete `ou=People,dc=example,dc=com` unless you send a subtree delete request control.

[1]The LDAP bind operation can potentially result in an update. Some directory servers can be configured to write time stamps in order to track successful or failed binds for password policy reasons.

As a rule, your client application must be authorized to create, update, and delete directory data. Therefore to prepare to change directory data, you first get a connection, and then bind on that connection as a user who is authorized to make the changes you plan to request.

8.2. Adding Directory Entries

The `Connection.add()` methods let you provide the entry to add as an `AddRequest`, an `Entry`, or as LDIF. If the changes to make are already expressed in LDIF, then you can also use `ChangeRecordReader`s, `ChangeRecord`s, and `ChangeRecordWriter`s to handle the changes.

The following excerpt demonstrates how to add a simple user entry under `ou=People,dc=example,dc=com`.

```
// An entry to add to the directory
Entry entry = new LinkedHashMapEntry("cn=Bob,ou=People,dc=example,dc=com")
    .addAttribute("cn", "Bob")
    .addAttribute("objectclass", "top")
    .addAttribute("objectclass", "person")
    .addAttribute("objectclass", "organizationalPerson")
    .addAttribute("objectclass", "inetOrgPerson")
    .addAttribute("mail", "subgenius@example.com")
    .addAttribute("sn", "Dobbs");

final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;
try {
    connection = factory.getConnection();
    // Bind as a user who has the right to add entries.
    connection.bind(adminDN, adminPwd);

    connection.add(entry);

} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

8.3. Modifying Directory Entry Attribute Values

The `Connection.modify()` methods let you add, replace, and delete attributes values on an entry. Either the modifications are expressed in LDIF, or you build a `ModifyRequest` to express the changes.

The following excerpt demonstrates how to replace one attribute value and to add another.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;
try {
    connection = factory.getConnection();
    // Bind as a user who has the right to modify entries.
    connection.bind(adminDN, adminPwd);

    // Here, entry is a user entry with DN cn=Bob,ou=People,dc=example,dc=com.
    Entry old = TreeMapEntry.deepCopyOfEntry(entry);
    entry = entry.replaceAttribute("mail", "spammer@example.com")
            .addAttribute("description", "I see the fnords.");
    ModifyRequest request = Entries.diffEntries(old, entry);

    connection.modify(request);

} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

8.4. Renaming Directory Entries

The `Connection.modifyDN()` methods serve to rename entries and to move them around.

The following excerpt demonstrates how to rename an entry.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;
try {
    connection = factory.getConnection();
    // Bind as a user who has the right to rename entries.
    connection.bind(adminDN, adminPwd);

    // Here, entryDN contains cn=Bob,ou=People,dc=example,dc=com.
    // The second argument is the new relative distinguished name.
    connection.modifyDN(entryDN, "cn=Ted");

} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

If you must move rather than rename entries, have a look at the methods for `ModifyDNRequest`. You can get a new request by using `Requests` static methods.

8.5. Deleting Directory Entries

The following excerpt demonstrates how to delete an entry with DN `cn=Ted,ou=People,dc=example,dc=com`.

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;
try {
    connection = factory.getConnection();
    // Bind as a user who has the right to delete entries.
    connection.bind(adminDN, adminPwd);

    connection.delete("cn=Ted,ou=People,dc=example,dc=com");

} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

If you must delete an entire branch of entries instead of a single leaf entry, build a `DeleteRequest` that includes the `SubtreeDeleteRequestControl`, as described in the section, *Subtree Delete Request Control*.

8.6. Updating Static Groups

Static groups enumerate user entries. Static groups can grow large. For an example, see the group entry at the end of big-group.ldif:

```
dn: cn=Static,ou=Groups,dc=example,dc=com
objectClass: top
objectClass: groupofnames
cn: Static
member: uid=user.0,ou=People,dc=example,dc=com
member: uid=user.1,ou=People,dc=example,dc=com
member: uid=user.2,ou=People,dc=example,dc=com
...
member: uid=user.10000,ou=People,dc=example,dc=com
```

To update a static group, you either add members or remove members.

The `UpdateGroup` example checks that the directory server supports the Permissive Modify control. With directory servers such as OpenDJ that support the LDAP Permissive Modify control, you can use the control to avoid having to determine whether a given member is already in the group before performing the operation. Instead you can simply request an add or a delete modification for the member.

Example 8.1. Updating a Group With Permissive Modify

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
```

```
Connection connection = null;
try {
    connection = factory.getConnection();

    Collection<String> controls =
            RootDSE.readRootDSE(connection).getSupportedControls();

    final String user = "cn=Directory Manager";
    final char[] password = "password".toCharArray();
    connection.bind(user, password);

    if (controls.contains(PermissiveModifyRequestControl.OID)) {

        final ModifyRequest request = Requests.newModifyRequest(groupDN)
                .addControl(PermissiveModifyRequestControl.newControl(true))
                .addModification(modType, "member", memberDN);
        connection.modify(request);

    } else {

        /* ... */

    }

    String op = (modType == ModificationType.ADD) ? "added to" : "deleted from";
    System.out.println("The entry with DN " + memberDN + " has been "
            + op + " the group with DN " + groupDN + ".");

} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

If the directory server does not support the Permissive Modify control, then the example checks whether the member is present in the group by using an LDAP compare operation. If a member to be added does not yet belong to the group, the example requests an add modification. If a member to be deleted does belong to the group, the example requests a delete modification.

Example 8.2. Updating a Group With Compare & Modify

```
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;
try {
    connection = factory.getConnection();

    Collection<String> controls =
            RootDSE.readRootDSE(connection).getSupportedControls();

    final String user = "cn=Directory Manager";
    final char[] password = "password".toCharArray();
    connection.bind(user, password);
    if (controls.contains(PermissiveModifyRequestControl.OID)) {

        /* ... */

    } else {

        System.out.println("Checking whether the entry with DN "
                + memberDN + " belongs to the group with DN " + groupDN
                + "...");
        final CompareRequest request =
                Requests.newCompareRequest(groupDN, "member", memberDN);
        CompareResult result = connection.compare(request);

        if (modType == ModificationType.ADD) {
            if (result.getResultCode() == ResultCode.COMPARE_FALSE) {
                System.out.println("Member does not yet belong to group."
                        + " Adding it...");
                final ModifyRequest addMember =
                        Requests.newModifyRequest(groupDN)
                            .addModification(modType, "member", memberDN);
                connection.modify(addMember);
            }
        }

        if (modType == ModificationType.DELETE) {
            if (result.getResultCode() == ResultCode.COMPARE_TRUE) {
                System.out.println("Member belongs to group."
                        + " Removing it...");
                final ModifyRequest delMember =
                        Requests.newModifyRequest(groupDN)
                            .addModification(modType, "member", memberDN);
                connection.modify(delMember);
            }
        }

    }

    String op = (modType == ModificationType.ADD) ? "added to" : "deleted from";
    System.out.println("The entry with DN " + memberDN + " has been "
            + op + " the group with DN " + groupDN + ".");

} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

You can change multiple member values with a single modification. The final argument of this form of the `ModifyRequest.addModification()` method takes a series of one or more values. So if you have multiple group members to add or delete, you can loop over your list to perform compare individual compare requests, then construct a single modify request to add or delete the group members. In other words, if you have three members to add, you can list the three member DNs as arguments of `addModification`.

```
String member1 = "uid=user1,ou=people,dc=example,dc=com";
String member2 = "uid=user1,ou=people,dc=example,dc=com";
String member3 = "uid=user1,ou=people,dc=example,dc=com";
final ModifyRequest addMember =
    Requests.newModifyRequest(groupDN)
        .addModification(modType, "member", member1, member2, member3);
connection.modify(addMember);
```

To try the example, download and import `big-group.ldif` into your directory server, and then run the sample. For example, if OpenDJ is set up to with directory manager as `cn=Directory Manager`, password `password` listening on `localhost` port `1389`, and you run the example with arguments `localhost 1389 cn=Static,ou=Groups,dc=example,dc=com uid=user.5150,ou=People,dc=example,dc=com del`, the resulting output is `The entry with DN uid=user.5150,ou=People,dc=example,dc=com has been deleted from the group with DN cn=Static,ou=Groups,dc=example,dc=com.`.

Chapter 9
Working With LDIF

OpenDJ LDAP SDK provides capabilities for working with LDAP Data Interchange Format (LDIF) content. This chapter demonstrates how to use those capabilities.

9.1. About LDIF

LDAP Data Interchange Format provides a mechanism to represent 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. When you read OpenDJ's external change log, you get changes expressed in LDIF.

LDIF uses base64 encoding to store values that are not safe for use in a text file, including values that represent binary objects like JPEG photos and X509 certificates, but also values that hold bits of LDIF, and values that end in white space. The description in the following LDIF holds, "Space at the end of the line " for example. Notice that continuation lines shown in the excerpt of the JPEG photo value start with spaces.

```
dn: uid=bjensen,ou=People,dc=example,dc=com
description:: U3BhY2UgYXQgdGhlIGVuZCBvZiB0aGUgbGluZSA=
uid: bjensen
jpegPhoto:: /9j/4AAQSkZJRgABAQEASABIAAD/4gxYSUNDX1BST0ZJTEUAAQEAAAxITGlubwIQAABt
 bnRyUkdCIFhZWiAHzgACAAkABgAxAABhY3NwTVNGVAAAAABJRUMgc1JHQgAAAAAAAAAAAAAAAAA9tY
 AAQAAAADTLUhQICAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
 ...
 Pxv8A8lh8J/8AXUfzr1qP/WSfWlzPlsZSi3VHqMA/WinUVB0n/9k=
facsimileTelephoneNumber: +1 408 555 1992
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
givenName: Barbara
cn: Barbara Jensen
cn: Babs Jensen
telephoneNumber: +1 408 555 1862
sn: Jensen
roomNumber: 0209
homeDirectory: /home/bjensen
ou: Product Development
ou: People
l: Cupertino
mail: bjensen@example.com
uidNumber: 1076
gidNumber: 1000
```

LDIF can serve to describe not only entries with their attributes but also changes to entries. For example, you can express adding a JPEG photo to Babs Jensen's entry as follows.

```
dn: uid=bjensen,ou=people,dc=example,dc=com
changetype: modify
add: jpegPhoto
jpegPhoto:< file:///tmp/opendj-logo.jpg
```

You can also replace and delete attribute values. Notice the dash, `-`, used to separate changes.

```
dn: uid=bjensen,ou=people,dc=example,dc=com
changetype: modify
replace: roomNumber
roomNumber: 1234
-
delete: description
-
delete: jpegPhoto
```

LDIF also allows `changetype`s of `add` to create entries, `delete` to remove entries, and `modrdn` to rename entries.

For more examples, see the LDIF specification, RFC 2849.

9.2. Reading LDIF

OpenDJ LDAP SDK provides `ChangeRecordReader`s to read requests to modify directory data, and `EntryReader`s to read entries from a data source such as a file or other source. Both of these are interfaces.

- The `ConnectionEntryReader` class offers methods to iterate through entries and references returned by a search.
- The `LDIFChangeRecordReader` and `LDIFEntryReader` classes offer methods to handle LDIF as strings or from an input stream.

  Both classes give you some methods to filter content. You can also use the `LDIF` static methods to filter content.

The following short excerpt shows a reader that takes LDIF change records from standard input.

```
InputStream ldif = System.in;
final LDIFChangeRecordReader reader = new LDIFChangeRecordReader(ldif);
```

9.3. Writing LDIF

`ChangeRecordWriter`s let you write requests to modify directory data, whereas `EntryWriter`s let you write entries to a file or an output stream. Both of these are interfaces.

- The `ConnectionChangeRecordWriter` and `ConnectionEntryWriter` classes let you write directly to a connection to the directory.
- The `LDIFChangeRecordWriter` and `LDIFEntryWriter` classes let you write to a file or other output stream. Both classes offer methods to filter content.

The following excerpt shows a writer pushing LDIF changes to a directory server.

```
final LDIFChangeRecordReader reader = new LDIFChangeRecordReader(ldif);
final LDAPConnectionFactory factory = new LDAPConnectionFactory(host, port);
Connection connection = null;

try {
    connection = factory.getConnection();
    connection.bind(userDN, password.toCharArray());

    final ConnectionChangeRecordWriter writer =
            new ConnectionChangeRecordWriter(connection);
    while (reader.hasNext()) {
        ChangeRecord changeRecord = reader.readChangeRecord();
        writer.writeChangeRecord(changeRecord);
    }
} catch (final ErrorResultException e) {
    System.err.println(e.getMessage());
    System.exit(e.getResult().getResultCode().intValue());
    return;
} catch (final IOException e) {
    System.err.println(e.getMessage());
    System.exit(ResultCode.CLIENT_SIDE_LOCAL_ERROR.intValue());
    return;
} finally {
    if (connection != null) {
        connection.close();
    }
}
```

Chapter 10
Working With Controls

This chapter demonstrates how to use LDAP controls.

10.1. About LDAP Controls

Controls provide a mechanism whereby the semantics and arguments of existing LDAP operations may be extended. One or more controls may be attached to a single LDAP message. A control only affects the semantics of the message it is attached to. Controls sent by clients are termed *request controls*, and those sent by servers are termed *response controls*.

10.2. Determining Supported Controls

For OpenDJ, the controls supported are listed in the *Administration Guide* appendix, *LDAP Controls* in the *Administration Guide*. You can access the list of OIDs for supported LDAP controls by reading the `supportedControl` attribute of the root DSE.

```
$ ldapsearch
 --baseDN ""
 --searchScope base
 --port 1389
 "(objectclass=*)" supportedControl
dn:
supportedControl: 1.2.826.0.1.3344810.2.3
supportedControl: 1.2.840.113556.1.4.1413
supportedControl: 1.2.840.113556.1.4.319
supportedControl: 1.2.840.113556.1.4.473
supportedControl: 1.2.840.113556.1.4.805
supportedControl: 1.3.6.1.1.12
supportedControl: 1.3.6.1.1.13.1
supportedControl: 1.3.6.1.1.13.2
supportedControl: 1.3.6.1.4.1.26027.1.5.2
supportedControl: 1.3.6.1.4.1.42.2.27.8.5.1
supportedControl: 1.3.6.1.4.1.42.2.27.9.5.2
supportedControl: 1.3.6.1.4.1.42.2.27.9.5.8
supportedControl: 1.3.6.1.4.1.4203.1.10.1
supportedControl: 1.3.6.1.4.1.4203.1.10.2
supportedControl: 1.3.6.1.4.1.7628.5.101.1
supportedControl: 2.16.840.1.113730.3.4.12
supportedControl: 2.16.840.1.113730.3.4.16
supportedControl: 2.16.840.1.113730.3.4.17
supportedControl: 2.16.840.1.113730.3.4.18
supportedControl: 2.16.840.1.113730.3.4.19
supportedControl: 2.16.840.1.113730.3.4.2
supportedControl: 2.16.840.1.113730.3.4.3
supportedControl: 2.16.840.1.113730.3.4.4
supportedControl: 2.16.840.1.113730.3.4.5
supportedControl: 2.16.840.1.113730.3.4.9
```

The following excerpt shows couple of methods to check whether the directory server supports a control.

```

/**
 * Controls supported by the LDAP server.
 */
private static Collection<String> controls;

/**
 * Populate the list of supported LDAP control OIDs.
 *
 * @param connection
 *            Active connection to the LDAP server.
 * @throws ErrorResultException
 *             Failed to get list of controls.
 */
static void checkSupportedControls(Connection connection)
        throws ErrorResultException {
    controls = RootDSE.readRootDSE(connection).getSupportedControls();
}

/**
 * Check whether a control is supported. Call {@code checkSupportedControls}
 * first.
 *
 * @param control
 *            Check support for this control, provided by OID.
 * @return True if the control is supported.
 */
static boolean isSupported(final String control) {
    if (controls != null && !controls.isEmpty()) {
        return controls.contains(control);
    }
    return false;
}
```

10.3. Assertion Request Control

The LDAP assertion control lets you specify a condition that must be true in order for the operation you request to be processed normally. The following excerpt shows, for example, how you might check that no description exists on the entry before adding your description.

```
if (isSupported(AssertionRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";

    final ModifyRequest request =
            Requests.newModifyRequest(dn)
                .addControl(AssertionRequestControl.newControl(
                        true, Filter.valueOf("!(description=*)")))
                .addModification(ModificationType.ADD, "description",
                        "Created using LDAP assertion control");

    connection.modify(request);

    final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
    try {
        writer.writeEntry(connection.readEntry(dn, "description"));
        writer.close();
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ directory server supports the LDAP assertion control:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Created using LDAP assertion control
```

10.4. Authorization Identity Controls

The LDAP Authorization Identity Controls let you get the authorization identity established when you bind to the directory server. The following excerpt shows simple use of the controls.

```
if (isSupported(AuthorizationIdentityRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";
    final char[] pwd = "hifalutin".toCharArray();

    System.out.println("Binding as " + dn);
    final BindRequest request =
            Requests.newSimpleBindRequest(dn, pwd)
                .addControl(AuthorizationIdentityRequestControl.newControl(true));

    final BindResult result = connection.bind(request);
    try {
        final AuthorizationIdentityResponseControl control =
                result.getControl(AuthorizationIdentityResponseControl.DECODER,
                        new DecodeOptions());
        System.out.println("Authorization ID returned: "
                        + control.getAuthorizationID());
    } catch (final DecodeException e) {
        // Failed to decode the response control.
    }
}
```

OpenDJ directory server supports the LDAP Authorization Identity Controls:

```
Binding as uid=bjensen,ou=People,dc=example,dc=com
Authorization ID returned: dn:uid=bjensen,ou=People,dc=example,dc=com
```

10.5. Entry Change Notification Response Controls

When performing a persistent search, your application can retrieve information using this response control about why the directory server returned the entry. See the Internet-Draft on persistent searches for background information.

```
if (isSupported(PersistentSearchRequestControl.OID)) {
    final SearchRequest request =
            Requests.newSearchRequest(
                    "dc=example,dc=com", SearchScope.WHOLE_SUBTREE,
                    "(objectclass=inetOrgPerson)", "cn")
                    .addControl(PersistentSearchRequestControl.newControl(
                            true, true, true, // critical,changesOnly,returnECs
                            PersistentSearchChangeType.ADD,
                            PersistentSearchChangeType.DELETE,
                            PersistentSearchChangeType.MODIFY,
                            PersistentSearchChangeType.MODIFY_DN));

    final ConnectionEntryReader reader = connection.search(request);

    try {
        while (reader.hasNext()) {
            if (!reader.isReference()) {
                final SearchResultEntry entry = reader.readEntry();
                System.out.println("Entry changed: "
                        + entry.getName().toString());
                EntryChangeNotificationResponseControl control =
                        entry.getControl(
                                EntryChangeNotificationResponseControl.DECODER,
                                new DecodeOptions());

                PersistentSearchChangeType type = control.getChangeType();
                System.out.println("Change type: " + type.toString());
                if (type.equals(PersistentSearchChangeType.MODIFY_DN)) {
                    System.out.println("Previous DN: "
                            + control.getPreviousName().toString());
                }
                System.out.println("Change number: "
                        + control.getChangeNumber());
                System.out.println(); // Add a blank line.
           }
        }
    } catch (final DecodeException e) {
        // Failed to decode the response control.
    } catch (final ErrorResultIOException e) {
        // Request failed due to an IO problem.
    } catch (final SearchResultReferenceIOException e) {
        // Read a reference, rather than an entry.
    }
}
```

OpenDJ directory server supports persistent searches and the entry change notification response control. When another application renames Anne-Louise Barnes's entry, the sample code picks up information from the entry change notification response control:

```
Entry changed: uid=bdobbs,ou=People,dc=example,dc=com
Change type: modifyDN
Previous DN: uid=abarnes,ou=People,dc=example,dc=com
Change number: -1
```

In this case, `Change number: -1` because the server did not set a change number value. OpenDJ directory server does not set the change number value in the response control. If you need to track the order of changes with OpenDJ directory server, read the external change log instead of using the entry change notification response control.

10.6. GetEffectiveRights Request Control

Your application can attach the GetEffectiveRights request control to retrieve information about what the directory server permits a user to do. Use this control during a search to see permissions on the entries returned. See the Internet-Draft on the Access Control Model for LDAP for background.

```
if (isSupported(GetEffectiveRightsRequestControl.OID)) {
    final String authDN = "uid=kvaughan,ou=People,dc=example,dc=com";

    final SearchRequest request =
            Requests.newSearchRequest(
                    "dc=example,dc=com", SearchScope.WHOLE_SUBTREE,
                    "(uid=bjensen)", "cn", "aclRights", "aclRightsInfo")
                    .addControl(GetEffectiveRightsRequestControl.newControl(
                            true, authDN, "cn"));

    final ConnectionEntryReader reader = connection.search(request);
    final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
    try {
        while (reader.hasNext()) {
            if (!reader.isReference()) {
                final SearchResultEntry entry = reader.readEntry();
                writer.writeEntry(entry);
            }
        }
        writer.close();
    } catch (final ErrorResultIOException e) {
        // Request failed due to an IO problem.
    } catch (final SearchResultReferenceIOException e) {
        // Read a reference, rather than an entry.
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ SDK currently implements the request control, but not the response control. The results are shown as values of the `aclRights` and more verbose `aclRightsInfo` attributes.

```
dn: uid=bjensen,ou=People,dc=example,dc=com
aclRightsInfo;logs;attributeLevel;selfwrite_delete;cn: acl_summary(main)
 : access allowed(write) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com
 , distinguishedName) to (uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: allow all Admin group)
aclRightsInfo;logs;entryLevel;read: acl_summary(main): access allowed(read
 ) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, objectClass) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied) ( reason
 : evaluated allow , deciding_aci: Anonymous read-search access)
aclRightsInfo;logs;attributeLevel;proxy;cn: acl_summary(main)
 : access not allowed(proxy) on entry/attr(uid=bjensen,ou=People,dc=example,
 dc=com, cn) to (uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) (reason: no acis matched the subject )
aclRights;attributeLevel;cn: search:1,read:1,compare:1,write:1,selfwrite_add:1,
 selfwrite_delete:1,proxy:0
aclRightsInfo;logs;attributeLevel;write;cn: acl_summary(main): access allowed
 (write) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, cn) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: allow all Admin group)
aclRights;entryLevel: add:1,delete:1,read:1,write:1,proxy:0
aclRightsInfo;logs;attributeLevel;search;cn: acl_summary(main): access allowed(
 search) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, cn) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: Anonymous read-search access)
```

```
aclRightsInfo;logs;entryLevel;write: acl_summary(main): access allowed(write
 ) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, NULL) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: allow all Admin group)
aclRightsInfo;logs;attributeLevel;selfwrite_add;cn: acl_summary(main
 ): access allowed(write) on entry/attr(uid=bjensen,ou=People,dc=example,
 dc=com, distinguishedName) to (uid=kvaughan,ou=People,dc=example,dc=com) (
 not proxied) ( reason: evaluated allow , deciding_aci: allow all Admin group)
aclRightsInfo;logs;entryLevel;add: acl_summary(main): access allowed(add
 ) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, NULL) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: allow all Admin group)
aclRightsInfo;logs;attributeLevel;read;cn: acl_summary(main): access allowed(
 read) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, cn) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: Anonymous read-search access)
cn: Barbara Jensen
cn: Babs Jensen
aclRightsInfo;logs;entryLevel;proxy: acl_summary(main): access not allowed(
 proxy) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, NULL) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: no acis matched the subject )
aclRightsInfo;logs;attributeLevel;compare;cn: acl_summary(main): access allowed
 (compare) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, cn) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: Anonymous read-search access)
aclRightsInfo;logs;entryLevel;delete: acl_summary(main): access allowed(
 delete) on entry/attr(uid=bjensen,ou=People,dc=example,dc=com, NULL) to (
 uid=kvaughan,ou=People,dc=example,dc=com) (not proxied
 ) ( reason: evaluated allow , deciding_aci: allow all Admin group)
```

10.7. ManageDsaIT Request Control

The ManageDsaIT control, described in RFC 3296, *Named Subordinate References in LDAP Directories*, lets your application handle references and other special entries as normal entries. Use it when you want to read from or write to reference or special entry.

```
if (isSupported(ManageDsaITRequestControl.OID)) {
    final String dn = "dc=ref,dc=com";

    final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
    try {
        System.out.println("Referral without the ManageDsaIT control.");
        SearchRequest request = Requests.newSearchRequest(dn,
                SearchScope.SUBORDINATES, "(objectclass=*)", "");
        final ConnectionEntryReader reader = connection.search(request);
        while (reader.hasNext()) {
            if (reader.isReference()) {
                final SearchResultReference ref = reader.readReference();
                System.out.println("Reference: " + ref.getURIs().toString());
            }
        }

        System.out.println("Referral with the ManageDsaIT control.");
        request.addControl(ManageDsaITRequestControl.newControl(true));
        final SearchResultEntry entry = connection.searchSingleEntry(request);
        writer.writeEntry(entry);
        writer.close();
    } catch (final ErrorResultIOException e) {
        // Request failed due to an IO problem.
    } catch (final SearchResultReferenceIOException e) {
        // Read a reference, rather than an entry.
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ directory server supports the ManageDsaIT Request Control. To use the example entry create a new base DN, `dc=ref,dc=com` before you import the data:

```
Referral without the ManageDsaIT control.
Reference: [ldap:///dc=example,dc=com??sub?:///dc=example,dc=com??sub?]
Referral with the ManageDsaIT control.
dn: dc=references,dc=ref,dc=com
```

10.8. Matched Values Request Control

RFC 3876, *Returning Matched Values with the LDAPv3*, describes a control that lets your application pass a filter in a search request getting a multivalued attribute such that the directory server only returns attribute values that match the filter.

Barbara Jensen's entry contains two common name values, `Barbara Jensen` and `Babs Jensen`. The following excerpt retrieves only the latter.

```
if (isSupported(MatchedValuesRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";
    final SearchRequest request =
            Requests.newSearchRequest(dn, SearchScope.BASE_OBJECT,
                    "(objectclass=*)", "cn")
                    .addControl(MatchedValuesRequestControl.newControl(
                            true, "(cn=Babs Jensen)"));

    final SearchResultEntry entry = connection.searchSingleEntry(request);
    System.out.println("Reading entry with matched values request.");
    final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
    try {
        writer.writeEntry(entry);
        writer.close();
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ directory server supports the matched values request control.

```
Reading entry with matched values request.
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Babs Jensen
```

10.9. Password Expired Response Control

A directory server can return the Password Expired Response Control, described in the Internet-Draft *Password Policy for LDAP Directories*, when a bind fails because the password has expired. In order to see this, you must configure the directory to expire Barbara Jensen's password.

```
if (isSupported(PasswordExpiredResponseControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";
    final char[] pwd = "hifalutin".toCharArray();

    try {
        connection.bind(dn, pwd);
    } catch (final ErrorResultException e) {
        final Result result = e.getResult();
        try {
            final PasswordExpiredResponseControl control =
                    result.getControl(PasswordExpiredResponseControl.DECODER,
                            new DecodeOptions());
            if (!(control == null) && control.hasValue()) {
                System.out.println("Password expired for " + dn);
            }
        } catch (final DecodeException de) {
            // Failed to decode the response control.
        }
    }
}
```

OpenDJ directory server supports the Password Expired Response Control. To obtain the following output from the excerpt, you can change the default password policy configuration to set a short maximum password age, change Barbara Jensen's password, and wait for it to expire. See the OpenDJ *Administration Guide* procedure explaining how *To Adjust the Default Password Policy* in the *Administration Guide* for an example of how to adjust the maximum password age.

```
Password expired for uid=bjensen,ou=People,dc=example,dc=com
```

10.10. Password Expiring Response Control

The Password Expiring Response Control, described in the Internet-Draft *Password Policy for LDAP Directories*, warns your application during a bind that the password used will soon expire.

```
if (isSupported(PasswordExpiringResponseControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";
    final char[] pwd = "hifalutin".toCharArray();

    final BindResult result = connection.bind(dn, pwd);
    try {
        final PasswordExpiringResponseControl control =
                result.getControl(PasswordExpiringResponseControl.DECODER,
                        new DecodeOptions());
        if (!(control == null) && control.hasValue()) {
            System.out.println("Password for " + dn + " expires in "
                    + control.getSecondsUntilExpiration() + " seconds.");
        }
    } catch (final DecodeException de) {
        // Failed to decode the response control.
    }
}
```

OpenDJ directory server supports the Password Expiring Response Control. To obtain the following output from the excerpt, you can change the default password policy configuration to set a maximum password age and a warning interval, change Barbara Jensen's password, and wait until you enter the warning interval before password expiration. See the OpenDJ *Administration Guide* procedure explaining how *To Adjust the Default Password Policy* in the *Administration Guide* for an example of how to adjust the maximum password age. Also set a short `password-expiration-warning-interval` value.

```
Password for uid=bjensen,ou=People,dc=example,dc=com
 expires in 237 seconds.
```

10.11. Password Policy Controls

The Behera Internet-Draft, *Password Policy for LDAP Directories*, describes Password Policy Request and Response Controls. You send the request control with a request to let the directory server know that your application can handle the response control. The directory server sends the response control on applicable operations to communicate warnings and errors.

```
if (isSupported(PasswordPolicyRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";
    final char[] pwd = "hifalutin".toCharArray();

    try {
        final BindRequest request = Requests.newSimpleBindRequest(dn, pwd)
                .addControl(PasswordPolicyRequestControl.newControl(true));

        final BindResult result = connection.bind(request);

        final PasswordPolicyResponseControl control =
                result.getControl(PasswordPolicyResponseControl.DECODER,
                        new DecodeOptions());
        if (!(control == null) && !(control.getWarningType() == null)) {
            System.out.println("Password policy warning "
                    + control.getWarningType().toString() + ", value "
                    + control.getWarningValue() + " for " + dn);
        }
    } catch (final ErrorResultException e) {
        final Result result = e.getResult();
        try {
            final PasswordPolicyResponseControl control =
                    result.getControl(PasswordPolicyResponseControl.DECODER,
                            new DecodeOptions());
            if (!(control == null)) {
                System.out.println("Password policy error "
                        + control.getErrorType().toString() + " for " + dn);
            }
        } catch (final DecodeException de) {
            // Failed to decode the response control.
        }
    } catch (final DecodeException e) {
        // Failed to decode the response control.
    }
}
```

OpenDJ directory server supports the Password Policy Controls. To obtain the output from the excerpt, you can change the default password policy configuration to set a maximum password age and a warning interval, change Barbara Jensen's password, and then run the example during the warning interval and after the password has expired. See the OpenDJ *Administration Guide* procedure explaining how *To Adjust the Default Password Policy* in the *Administration Guide* for an example of how to adjust the maximum password age. Also set a short `password-expiration-warning-interval` value.

For a warning:

```
Password policy warning timeBeforeExpiration, value 237 for
 uid=bjensen,ou=People,dc=example,dc=com
```

For an error:

```
Password policy error passwordExpired for
 uid=bjensen,ou=People,dc=example,dc=com
```

10.12. Permissive Modify Request Control

Microsoft defined a Permissive Modify Request Control that relaxes some constraints when your application performs a modify operation and tries to `add` an attribute that already exists, or to `delete` an attribute that does not exist.

```
if (isSupported(PermissiveModifyRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";

    final ModifyRequest request =
            Requests.newModifyRequest(dn)
                .addControl(PermissiveModifyRequestControl.newControl(true))
                .addModification(ModificationType.ADD, "uid", "bjensen");

    connection.modify(request);
    System.out.println("Permissive modify did not complain about "
            + "attempt to add uid: bjensen to " + dn + ".");
}
```

OpenDJ directory server supports the Permissive Modify Request Control:

```
Permissive modify did not complain about attempt to add
 uid: bjensen to uid=bjensen,ou=People,dc=example,dc=com.
```

10.13. Persistent Search Request Control

See Section 10.5, "Entry Change Notification Response Controls".

10.14. Post-Read Controls

RFC 4527, *LDAP Read Entry Controls*, describes the post-read controls that let your application get the content of an entry immediately after modifications are applied.

```
if (isSupported(PostReadRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";

    final ModifyRequest request =
            Requests.newModifyRequest(dn)
            .addControl(PostReadRequestControl.newControl(true, "description"))
            .addModification(ModificationType.REPLACE,
                    "description", "Using the PostReadRequestControl");

    final Result result = connection.modify(request);
    try {
        final PostReadResponseControl control =
                result.getControl(PostReadResponseControl.DECODER,
                        new DecodeOptions());
        final Entry entry = control.getEntry();

        final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
        writer.writeEntry(entry);
        writer.close();
    } catch (final DecodeException e) {
        // Failed to decode the response control.
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ directory server supports these controls:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Using the PostReadRequestControl
```

10.15. Pre-Read Controls

RFC 4527, *LDAP Read Entry Controls*, describes the pre-read controls that let your application get the content of an entry immediately before modifications are applied.

```
if (isSupported(PreReadRequestControl.OID)) {
    final String dn = "uid=bjensen,ou=People,dc=example,dc=com";

    final ModifyRequest request =
            Requests.newModifyRequest(dn)
            .addControl(PreReadRequestControl.newControl(true, "mail"))
            .addModification(
                    ModificationType.REPLACE, "mail", "modified@example.com");

    final Result result = connection.modify(request);
    try {
        final PreReadResponseControl control =
                result.getControl(PreReadResponseControl.DECODER,
                        new DecodeOptions());
        final Entry entry = control.getEntry();

        final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
        writer.writeEntry(entry);
        writer.close();
    } catch (final DecodeException e) {
        // Failed to decode the response control.
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ directory server supports these controls:

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

10.16. Proxied Authorization Request Controls

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

```
if (isSupported(ProxiedAuthV2RequestControl.OID)) {
    final String bindDN = "cn=My App,ou=Apps,dc=example,dc=com";
    final String targetDn = "uid=bjensen,ou=People,dc=example,dc=com";
    final String authzId = "dn:uid=kvaughan,ou=People,dc=example,dc=com";

    final ModifyRequest request =
            Requests.newModifyRequest(targetDn)
            .addControl(ProxiedAuthV2RequestControl.newControl(authzId))
            .addModification(ModificationType.REPLACE, "description",
                    "Done with proxied authz");

    connection.bind(bindDN, "password".toCharArray());
    connection.modify(request);
    final Entry entry = connection.readEntry(targetDn, "description");

    final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
    try {
        writer.writeEntry(entry);
        writer.close();
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ supports proxied authorization, and the example works with the sample data:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Done with proxied authz
```

10.17. Server-Side Sort Controls

The server-side sort controls are described in RFC 2891, *LDAP Control Extension for Server Side Sorting of Search Results*. If possible, sort on the client side instead to reduce load on the server. If not, then you can request a server-side sort.

```
static void useServerSideSortRequestControl(Connection connection)
        throws ErrorResultException {
    if (isSupported(ServerSideSortRequestControl.OID)) {
        final SearchRequest request =
                Requests.newSearchRequest("ou=People,dc=example,dc=com",
                        SearchScope.WHOLE_SUBTREE, "(sn=Jensen)", "cn")
                        .addControl(ServerSideSortRequestControl.newControl(
                                true, new SortKey("cn")));

        final SearchResultHandler resultHandler = new MySearchResultHandler();
        final Result result = connection.search(request, resultHandler);

        try {
            final ServerSideSortResponseControl control =
                    result.getControl(ServerSideSortResponseControl.DECODER,
                            new DecodeOptions());
            if (control != null && control.getResult() == ResultCode.SUCCESS) {
                System.out.println("# Entries are sorted.");
            } else {
                System.out.println("# Entries not necessarily sorted");
            }
        } catch (final DecodeException e) {
            // Failed to decode the response control.
        }
    } else {
        System.out.println("ServerSideSortRequestControl not supported");
    }
}

private static class MySearchResultHandler implements SearchResultHandler {

    @Override
    public void handleErrorResult(ErrorResultException error) {
        // Ignore.
    }

    @Override
    public void handleResult(Result result) {
        // Ignore.
    }

    @Override
    public boolean handleEntry(SearchResultEntry entry) {
        final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
        try {
            writer.writeEntry(entry);
            writer.flush();
        } catch (final IOException e) {
            // The writer could not write to System.out.
        }
        return true;
    }

    @Override
    public boolean handleReference(SearchResultReference reference) {
        System.out.println("Got a reference: " + reference.toString());
        return false;
    }
}
```

OpenDJ directory server supports server-side sorting:

```
dn: uid=ajensen,ou=People,dc=example,dc=com
cn: Allison Jensen

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

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

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

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

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

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

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

dn: uid=tjensen,ou=People,dc=example,dc=com
cn: Ted Jensen

# Entries are sorted.
```

10.18. Simple Paged Results Control

RFC 2696, *LDAP Control Extension for Simple Paged Results Manipulation*, defines a control for simple paging of search results that works with a cookie mechanism.

```
if (isSupported(SimplePagedResultsControl.OID)) {
    ByteString cookie = ByteString.empty();
    SearchRequest request;
    final SearchResultHandler resultHandler = new MySearchResultHandler();
    Result result;

    int page = 1;
    do {
        System.out.println("# Simple paged results: Page " + page);

        request =
                Requests.newSearchRequest("dc=example,dc=com",
                        SearchScope.WHOLE_SUBTREE, "(sn=Jensen)", "cn")
                        .addControl(SimplePagedResultsControl.newControl(
                                true, 3, cookie));

        result = connection.search(request, resultHandler);
        try {
            SimplePagedResultsControl control =
                    result.getControl(SimplePagedResultsControl.DECODER,
                            new DecodeOptions());
            cookie = control.getCookie();
        } catch (final DecodeException e) {
            // Failed to decode the response control.
        }

        ++page;
    } while (cookie.length() != 0);
}
```

OpenDJ directory server supports getting simple paged results:

```
# Simple paged results: Page 1
dn: uid=ajensen,ou=People,dc=example,dc=com
cn: Allison Jensen

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

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

# Simple paged results: Page 2
dn: uid=gjensen,ou=People,dc=example,dc=com
cn: Gern Jensen

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

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

# Simple paged results: Page 3
dn: uid=rjense2,ou=People,dc=example,dc=com
cn: Randy Jensen

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

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

10.19. Subentries Request Control

RFC 3672, *Subentries in LDAP*, describes subentries and also the subentries request control. When you perform a search without the control and visibility set to `TRUE`, subentries are only visible in searches with `SearchScope.BASE_OBJECT`.

```
if (isSupported(SubentriesRequestControl.OID)) {
    final SearchRequest request =
            Requests.newSearchRequest("dc=example,dc=com",
                        SearchScope.WHOLE_SUBTREE,
                        "cn=*Class of Service", "cn", "subtreeSpecification")
                    .addControl(SubentriesRequestControl.newControl(
                        true, true));

    final ConnectionEntryReader reader = connection.search(request);
    final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);
    try {
        while (reader.hasNext()) {
            if (reader.isEntry()) {
                final SearchResultEntry entry = reader.readEntry();
                writer.writeEntry(entry);
            }
        }
        writer.close();
    } catch (final ErrorResultIOException e) {
        // Request failed due to an IO problem.
    } catch (final SearchResultReferenceIOException e) {
        // Read a reference, rather than an entry.
    } catch (final IOException e) {
        // The writer could not write to System.out.
    }
}
```

OpenDJ directory server supports the control.

```
dn: cn=Bronze Class of Service,dc=example,dc=com
cn: Bronze Class of Service
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=
 bronze)" }

dn: cn=Silver Class of Service,dc=example,dc=com
cn: Silver Class of Service
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=
 silver)" }

dn: cn=Gold Class of Service,dc=example,dc=com
cn: Gold Class of Service
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=
 gold)" }
```

10.20. Subtree Delete Request Control

The subtree delete request control, described in the Internet-Draft *Tree Delete Control*, lets your application delete an entire branch of entries starting with the entry you target for deletion.

```
if (isSupported(SubtreeDeleteRequestControl.OID)) {

    final String dn = "ou=Apps,dc=example,dc=com";
    final DeleteRequest request =
            Requests.newDeleteRequest(dn)
                    .addControl(SubtreeDeleteRequestControl.newControl(true));

    final Result result = connection.delete(request);
    if (result.isSuccess()) {
        System.out.println("Successfully deleted " + dn
                + " and all entries below.");
    } else {
        System.out.println("Result: " + result.getDiagnosticMessage());
    }
}
```

OpenDJ directory server supports the subtree delete control:

```
Successfully deleted ou=Apps,dc=example,dc=com and all entries below.
```

10.21. Virtual List View Controls

The virtual list view controls are intended to be used by applications that let users browse lists of directory entries. The Internet-Draft *LDAP Extensions for Scrolling View Browsing of Search Results* describes the controls. The virtual list view request control is used in conjunction with the server-side sort control such that the subset of entries the directory server returns from a search are a window into the full sorted list.

```
if (isSupported(VirtualListViewRequestControl.OID)) {
    ByteString contextID = ByteString.empty();

    // Add a window of 2 entries on either side of the first sn=Jensen entry.
    final SearchRequest request =
            Requests.newSearchRequest("ou=People,dc=example,dc=com",
                    SearchScope.WHOLE_SUBTREE, "(sn=*)", "sn", "givenName")
                    .addControl(ServerSideSortRequestControl.newControl(
                            true, new SortKey("sn")))
                    .addControl(
                            VirtualListViewRequestControl.newAssertionControl(
                                    true,
                                    ByteString.valueOf("Jensen"),
                                    2, 2, contextID));

    final SearchResultHandler resultHandler = new MySearchResultHandler();
    final Result result = connection.search(request, resultHandler);

    try {
        final ServerSideSortResponseControl sssControl =
                result.getControl(ServerSideSortResponseControl.DECODER,
                        new DecodeOptions());
        if (sssControl != null && sssControl.getResult() == ResultCode.SUCCESS){
            System.out.println("# Entries are sorted.");
        } else {
            System.out.println("# Entries not necessarily sorted");
        }

        final VirtualListViewResponseControl vlvControl =
                result.getControl(VirtualListViewResponseControl.DECODER,
                        new DecodeOptions());
        System.out.println("# Position in list: "
                + vlvControl.getTargetPosition() + "/"
                + vlvControl.getContentCount());
    } catch (final DecodeException e) {
        // Failed to decode the response control.
    }
}
```

OpenDJ directory server supports the virtual list view controls. In order to set up OpenDJ directory server to produce the following output with the example code, use OpenDJ Control Panel > Manage Indexes > New VLV Index... to set up a virtual list view index for people by last name, using the filter `(|(givenName=*)(sn=*))`, and sorting first by surname, `sn`, in ascending order, then by given name also in ascending order.

```
dn: uid=skellehe,ou=People,dc=example,dc=com
givenName: Sue
sn: Kelleher

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

dn: uid=ajensen,ou=People,dc=example,dc=com
givenName: Allison
sn: Jensen

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

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

# Entries are sorted.
# Position in list: 92/150
```

10.22. Using a Generic Control

OpenDJ LDAP SDK supports many controls, but you might still need to work with additional controls. If so, then in some cases you can use the `GenericControl` class when adding the control to your request.

For example, the Microsoft LDAP Server Notification Control with OID `1.2.840.113556.1.4.528` can be used to register a change notification request for a search on Microsoft Active Directory. You can use a `GenericControl.newControl()` static method to add the request control to your search.

```
        final SearchScope scope = SearchScope.WHOLE_SUBTREE;
        final String filter = "(objectclass=*)";
        final String[] attributes = {
            "objectclass", "objectGUID", "isDeleted", "uSNChanged"
        };

        // Create an LDIF writer which will write the search results to stdout.
        final LDIFEntryWriter writer = new LDIFEntryWriter(System.out);

        // Connect and bind to the server.
        final LDAPConnectionFactory factory =
                new LDAPConnectionFactory(hostName, port);
        Connection connection = null;

        try {
            connection = factory.getConnection();
            connection.bind(userName, password.toCharArray());

            // Read the entries and output them as LDIF.
            final SearchRequest request =
                    Requests
                            .newSearchRequest(baseDN, scope, filter, attributes)
                            .addControl(
                                    GenericControl
                                            .newControl(
                                                    "1.2.840.113556.1.4.528",
                                                    true));
            final ConnectionEntryReader reader = connection.search(request);
            while (reader.hasNext()) {
                if (!reader.isReference()) {
                    final SearchResultEntry entry = reader.readEntry();
                    writer.writeComment("Search result entry: "
                            + entry.getName().toString());
                    writer.writeEntry(entry);
                    writer.flush();
                } else {
                    final SearchResultReference ref = reader.readReference();

                    // Got a continuation reference.
                    writer.writeComment("Search result reference: "
                            + ref.getURIs().toString());
                }
            }
        }
```

When you run the search against Active Directory and then create, update, and delete a new user, in this example `CN=New User,CN=Users,DC=ad,DC=example,DC=com`, Active Directory notifies you of changes to directory data.

```
# Search result entry: CN=RID Set,CN=WIN2008R2641,OU=Domain Controllers,
 DC=ad,DC=example,DC=com
dn: CN=RID Set,CN=WIN2008R2641,OU=Domain Controllers,DC=ad,DC=example,DC=com
objectClass: top
objectClass: rIDSet
objectGUID:: 178zQQic3EOoBOB1j2QVgQ==
uSNChanged: 12446

# Search result entry: CN=New User,CN=Users,DC=ad,DC=example,DC=com
dn: CN=New User,CN=Users,DC=ad,DC=example,DC=com
objectClass: top
```

```
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectGUID:: 7XE/OoJdFEqAegwAi2eNlA==
uSNChanged: 12753

# Search result entry: CN=New User,CN=Users,DC=ad,DC=example,DC=com
dn: CN=New User,CN=Users,DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectGUID:: 7XE/OoJdFEqAegwAi2eNlA==
uSNChanged: 12755

# Search result entry: CN=New User,CN=Users,DC=ad,DC=example,DC=com
dn: CN=New User,CN=Users,DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectGUID:: 7XE/OoJdFEqAegwAi2eNlA==
uSNChanged: 12757

# Search result entry: CN=New User,CN=Users,DC=ad,DC=example,DC=com
dn: CN=New User,CN=Users,DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectGUID:: 7XE/OoJdFEqAegwAi2eNlA==
uSNChanged: 12758

# Search result entry: CN=New User\0ADEL:3a3f71ed-5d82-4a14-807a-0c008b678d94,
# CN=Deleted Objects,DC=ad,DC=example,DC=com
dn: CN=New User\0ADEL:3a3f71ed-5d82-4a14-807a-0c008b678d94,CN=Deleted Objects,
 DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
objectGUID:: 7XE/OoJdFEqAegwAi2eNlA==
isDeleted: TRUE
uSNChanged: 12759
```

The `GenericControl` class is useful with controls that do not require you to encode complex request values, or decode complex response values. If the control you want to you requires complex encoding or decoding, you might have to implement `org.forgerock.opendj.ldap.controls.Control`.

Chapter 11
Working With Extended Operations

This chapter demonstrates how to use LDAP extended operations.

11.1. About LDAP Extended Operations

Extended operations allow additional operations to be defined for services not already available in the protocol

11.2. Determining Supported Extended Operations

For OpenDJ, the extended operations supported are listed in the *Administration Guide* appendix, *LDAP Extended Operations* in the *Administration Guide*. You can access the list of OIDs for supported LDAP controls by reading the `supportedExtension` attribute of the root DSE.

```
$ ldapsearch
 --baseDN ""
 --searchScope base
 --port 1389
 "(objectclass=*)" supportedExtension
dn:
supportedExtension: 1.3.6.1.1.8
supportedExtension: 1.3.6.1.4.1.26027.1.6.1
supportedExtension: 1.3.6.1.4.1.26027.1.6.2
supportedExtension: 1.3.6.1.4.1.26027.1.6.3
supportedExtension: 1.3.6.1.4.1.4203.1.11.1
supportedExtension: 1.3.6.1.4.1.4203.1.11.3
supportedExtension: 1.3.6.1.4.1.1466.20037
```

The following excerpt shows code to check for supported extended operations.

```

/**
 * Controls supported by the LDAP server.
 */
private static Collection<String> extendedOperations;

/**
 * Populate the list of supported LDAP extended operation OIDs.
 *
 * @param connection
 *            Active connection to the LDAP server.
 * @throws ErrorResultException
 *             Failed to get list of extended operations.
 */
static void checkSupportedExtendedOperations(Connection connection)
        throws ErrorResultException {
    extendedOperations = RootDSE.readRootDSE(connection)
            .getSupportedExtendedOperations();
}

/**
 * Check whether an extended operation is supported. Call
 * {@code checkSupportedExtendedOperations} first.
 *
 * @param extendedOperation
 *            Check support for this extended operation, provided by OID.
 * @return True if the control is supported.
 */
static boolean isSupported(final String extendedOperation) {
    if (extendedOperations != null && !extendedOperations.isEmpty()) {
        return extendedOperations.contains(extendedOperation);
    }
    return false;
}
```

11.3. Cancel Extended Operation

RFC 3909, *LDAP Cancel Operation*, defines an extended operation that lets you cancel an operation in progress and get an indication of the outcome.

The Cancel extended request uses the request ID of operation you want to cancel, and so therefore works with asynchronous searches and updates. Depending on the delay between your application sending the Cancel request and the directory server receiving the request, the server might have already finished processing the original request before it receives your Cancel request.

You can add a Cancel extended request for example to stop handling entries returned from a search if the directory server returns more entries than you want.

```
private static final CountDownLatch COMPLETION_LATCH = new CountDownLatch(1);
private static final CountDownLatch CANCEL_LATCH = new CountDownLatch(1);
private static final LDIFEntryWriter WRITER = new LDIFEntryWriter(System.out);

static int requestID;
static int entryCount = 0;

// The requestID is obtained from the future result of the asynchronous search.
// For more context see the example, SearchAsync.java.

private static final class SearchResultHandlerImpl
        implements SearchResultHandler {

    @Override
    public synchronized boolean handleEntry(final SearchResultEntry entry) {
        try {
            // Cancel the search if it returns too many results.
            if (entryCount < 10) {
                WRITER.writeComment("Search result entry: "
                        + entry.getName().toString());
                WRITER.writeEntry(entry);
                ++entryCount;
            } else {
                CancelExtendedRequest request =
                        Requests.newCancelExtendedRequest(requestID);
                connection.extendedRequestAsync(
                        request, null, new CancelResultHandlerImpl());
                return false;
            }
        } catch (final IOException e) {
            System.err.println(e.getMessage());
            resultCode = ResultCode.CLIENT_SIDE_LOCAL_ERROR.intValue();
            COMPLETION_LATCH.countDown();
            return false;
        }
        return true;
    }
    ...
}

private static final class CancelResultHandlerImpl
        implements ResultHandler<ExtendedResult> {

    @Override
    public void handleErrorResult(final ErrorResultException error) {
        System.err.println("Cancel request failed with result code: "
                + error.getResult().getResultCode().intValue());
        CANCEL_LATCH.countDown();
    }

    @Override
    public void handleResult(final ExtendedResult result) {
        System.err.println("Cancel request succeeded");
        CANCEL_LATCH.countDown();
    }

}
```

OpenDJ directory server supports the cancel operation. If OpenDJ directory server manages to return all entries in `Example.ldif` before it receives the Cancel extended request, you can see the Cancel request fail because the request ID refers to the search, which is no longer in progress. Try adding a new base DN using OpenDJ control panel and adding the default 2000 generated entries to ensure more search results. For example if `dc=example,dc=org` contains 2000 generated entries, and the `SearchAsync` example is run with the arguments `sub objectclass=* cn` for scope, filter, and attributes respectively, then the example produces something like the following output:

```
Canceled: Processing on this operation was terminated as a result of receiving
 a cancel request (message ID 3)
# Search result entry: dc=example,dc=org
dn: dc=example,dc=org

# Search result entry: ou=People,dc=example,dc=org
dn: ou=People,dc=example,dc=org

# Search result entry: uid=user.0,ou=People,dc=example,dc=org
dn: uid=user.0,ou=People,dc=example,dc=org
cn: Aaccf Amar

...

Cancel request succeeded
```

11.4. Password Modify Extended Operation

RFC 3062, *LDAP Password Modify Extended Operation*, defines an extended operation for modifying user passwords that does not depend on the authentication identity, nor on the way passwords are stored.

```
if (isSupported(PasswordModifyExtendedRequest.OID)) {
    final String userIdentity = "u:scarter";
    final char[] oldPassword = "sprain".toCharArray();
    final char[] newPassword = "secret12".toCharArray();

    final PasswordModifyExtendedRequest request =
            Requests.newPasswordModifyExtendedRequest()
                .setUserIdentity(userIdentity)
                .setOldPassword(oldPassword)
                .setNewPassword(newPassword);

    final PasswordModifyExtendedResult result =
            connection.extendedRequest(request);
    if (result.isSuccess()) {
        System.out.println("Changed password for " + userIdentity);
    } else {
        System.err.println(result.getDiagnosticMessage());
    }
}
```

OpenDJ directory server supports the password modify operation.

```
Changed password for u:scarter
```

11.5. Start TLS Extended Operation

Use Start TLS when setting up your connection to protect what your application sends to and receives from the directory server. For an example, read the section on *Start TLS & SSL Authentication*.

11.6. Who am I? Extended Operation

RFC 4532, *LDAP "Who am I?" Operation*, defines an extended operation that lets your application determine the current authorization ID.

```
if (isSupported(WhoAmIExtendedRequest.OID)) {

    final String name = "uid=bjensen,ou=People,dc=example,dc=com";
    final char[] password = "hifalutin".toCharArray();

    final Result result = connection.bind(name, password);
    if (result.isSuccess()) {

        final WhoAmIExtendedRequest request =
                Requests.newWhoAmIExtendedRequest();
        final WhoAmIExtendedResult extResult =
                connection.extendedRequest(request);

        if (extResult.isSuccess()) {
            System.out.println("Authz ID: "  + extResult.getAuthorizationID());
        }
    }
}
```

OpenDJ directory server supports the "Who am I?" operation.

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

Chapter 12
Internationalizing Applications

When you internationalize your application — adapting your application for use in different languages and regions — how much you do depends on what you must later localize. Directory servers often support localized user data. OpenDJ directory server supports use of the locales provided by your Java installation, and also supports many language subtypes, for example.

Therefore if your application is not end user facing and the administrators managing your application all use the same language as you do, you might be content to use language subtypes in LDAP filters, as described in the section on *Working With Search Filters*.

For end user facing applications where you must return localized messages, and for applications where administrators need localized log messages, you can use the ForgeRock I18N Framework.

Chapter 13
Writing a Simple LDAP Proxy

This chapter considers a simple LDAP proxy that forwards requests to one or more remote directory servers. Although the implementation is intended as an example, it does demonstrate use of the asynchronous API, load balancing, and connection pooling.

The Proxy example sets up connections pools with load balancing to the directory servers. It passes the connection factories to a `ProxyBackend` that handles the requests passed back to the directory servers. It also sets up an LDAP listener to receive incoming connections from clients of the Proxy.

The `ProxyBackend` uses separate connection factories, one for bind operations, the other for other operations. It uses the proxied authorization control to ensure operations are performed using the bind identity for the operation.

The `ProxyBackend`'s function is to handle each client request, encapsulating the result handlers that allow it to deal with each basic operation. It authenticates to the directory server to check incoming credentials, and adds the proxied authorization control to requests other than binds. The `ProxyBackend` handles all operations using asynchronous connections and methods.

13.1. Connection Pooling

As shown in the Proxy example, the `Connections.newFixedConnectionPool()` returns a connection pool of the maximum size you specify.

```
final List<ConnectionFactory> factories = new LinkedList<~>();

factories.add(Connections.newFixedConnectionPool(Connections
        .newAuthenticatedConnectionFactory(Connections
                .newHeartBeatConnectionFactory(new LDAPConnectionFactory(
                        remoteAddress, remotePort)),
                        Requests.newSimpleBindRequest(proxyDN,
                                proxyPassword.toCharArray())),
                                Integer.MAX_VALUE));
```

Connections are returned to the pool when you `close()` them. Notice that `Connections` also provides methods to return `ConnectionFactory`s with a heart beat check on connections provided by the factory, and connection factories that authenticate connections before returning them.

Connections in the pool are intended for reuse. The Proxy gets an authenticated connection, which is a connection where the OpenDJ LDAP SDK passes a bind request immediately when getting the connection. The Proxy then uses proxied authorization to handle the identity from the client requesting the operation. As a rule, either handle binds separately and use proxied authorization as in the Proxy example, or else make sure that the first operation on a connection retrieved from the pool is a bind that correctly authenticates the user currently served by the connection.

When you `close()` a connection from the pool, the OpenDJ LDAP SDK does not perform an `unbind()`. This is why you must be careful about how you manage authentication on connections from a pool.

13.2. Load Balancing & Failover

The `Connections.newLoadBalancer()` method returns a load balancer based on the algorithm you choose. Algorithms include both round robin for equitably sharing load across local directory servers, and also failover usually used for switching automatically from an unresponsive server group to an alternative server group. The algorithms take collections of connection factories, such as those that you set up for connection pooling.

The following excerpt shows how to set up round robin load balancing across directory servers.

```
final List<ConnectionFactory> factories = new LinkedList<ConnectionFactory>();

// Set up a ConnectionFactory for each directory server in the pool as shown in
// the previous example, and then set up a load balancer.

final RoundRobinLoadBalancingAlgorithm algorithm =
        new RoundRobinLoadBalancingAlgorithm(factories);

final ConnectionFactory factory = Connections.newLoadBalancer(algorithm);
```

With multiple pools of directory servers, for example in a deployment across multiple data centers, also use fail over load balancing. Fail over load balancing directs all requests to the first (preferred) pool of servers until problems are encountered with the connections to that pool. Then it fails over to the next pool in the list. Therefore in each data center you can set up round robin load balancing, and then set up fail over load balancing across data centers.

```
// localFactory:  ConnectionFactory to servers in the local data center
// remoteFactory: ConnectionFactory for servers in a remote data center
// localFactory and remoteFactory use round robin load balancing "internally".

final List<ConnectionFactory> factories =
        Arrays.asList(localFactory, remoteFactory);

final FailoverLoadBalancingAlgorithm algorithm =
        new FailoverLoadBalancingAlgorithm(factories);

final ConnectionFactory factory = Connections.newLoadBalancer(algorithm);
```

The algorithms also include constructors that let you adjust timeouts and so forth.

13.3. Listening For & Handling Client Connections

You create an `LDAPListener` to handle incoming client connections. The `LDAPListener` takes a connection handler that deals with the connections, in this case connections back to the directory servers handling client requests.

```
final LDAPListenerOptions options = new LDAPListenerOptions().setBacklog(4096);
LDAPListener listener = null;
try {
    listener = new LDAPListener(localAddress, localPort, connectionHandler,
            options);
    System.out.println("Press any key to stop the server...");
    System.in.read();
} catch (final IOException e) {
    System.out.println("Error listening on " + localAddress + ":" + localPort);
    e.printStackTrace();
} finally {
    if (listener != null) {
        listener.close();
    }
}
```

You get a `ServerConnectionFactory` to handle requests coming from clients. The `ServerConnectionFactory` takes a request handler that deals with the incoming client requests. The request handler implements handlers for all supported operations. The Proxy example implements a `ProxyBackend` to handle requests. The `ProxyBackend` sends the requests on to the backend directory servers and routes the results returned back to client applications.

```
final ProxyBackend backend = new ProxyBackend(factory, bindFactory);
final ServerConnectionFactory<LDAPClientContext, Integer> connectionHandler =
        Connections.newServerConnectionFactory(backend);
```

See the Proxy example code for details about the `ProxyBackend` implementation.

13.4. DN & Attribute Rewriting

Suppose you have a client application that expects a different attribute name, such as `fullname` for a standard attribute like `cn` (common name), and that expects a distinguished name (DN) suffix different from what is stored in the directory. If you cannot change the application, one possible alternative is a proxy layer that does DN and attribute rewriting.[1]

[1]Some servers, such as OpenDJ directory server, can do attribute rewriting without a proxy layer. See your directory server's documentation for details.

```
# A search accessing the directory server
$ ldapsearch -b dc=example,dc=com -p 1389 "(cn=Babs Jensen)" cn
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen

# The same search search accessing a proxy that rewrites requests and responses
$ ldapsearch -b o=example -p 8389 "(fullname=Babs Jensen)" fullname
dn: uid=bjensen,ou=People,o=example
fullname: Barbara Jensen
fullname: Babs Jensen
```

In the above output, the rewriter proxy listens on port 8389, connecting to a directory server listening on 1389. The directory server contains data from .

Tools Reference

You can find the tools under the `bin/` or `bat\` folder where you installed OpenDJ LDAP SDK toolkit as described in the procedure explaining how *To Install the Latest SDK & Tools*. For example, `/path/to/opendj-ldap-toolkit-2.6.0/bin`.

Table of Contents

Name

authrate — measure bind throughput and response time

Synopsis

`authrate` {options} [filter format string] [attributes...]

Description

This utility can be used to measure bind throughput and response time of a directory service using user-defined bind or search-then-bind operations.

Format strings may be used in the bind DN option as well as the authid and authzid SASL bind options. A search operation may be used to retrieve the bind DN by specifying the base DN and a filter. The retrieved entry DN will be appended as the last argument in the argument list when evaluating format strings.

Options

The following options are supported.

`-a, --dereferencePolicy {dereferencePolicy}`

Alias dereference policy ('never', 'always', 'search', or 'find')

Default value: never

`-b, --baseDN {baseDN}`

Base DN format string

`-c, --numConnections {numConnections}`

Number of connections

Default value: 1

`-e, --percentile {percentile}`

Calculate max response time for a percentile of operations

`-f, --keepConnectionsOpen`

Keep connections open

`-g, --argument {generator function or static string}`

Argument used to evaluate the Java style format strings in program parameters (Base DN, Search Filter). The set of all arguments provided form the the argument list in order. Besides static string arguments, they can be generated per iteration with the following functions:

"inc({filename})"

Consecutive, incremental line from file

"inc({min},{max})"

Consecutive, incremental number

"rand({filename})"

Random line from file

"rand({min},{max})"

Random number

"randstr({length},*charSet*)"

Random string of specified length and optionally from characters in the charSet string. A range of character can be specified with [start-end] charSet notation. If no charSet is specified, the default charSet of [A-Z][a-z][0-9] will be used.

`-i, --statInterval {statInterval}`

Display results each specified number of seconds

Default value: 5

`-I, --invalidPassword {invalidPassword}`

Percent of bind operations with simulated invalid password

Default value: 0

`-m, --maxIterations {maxIterations}`

Max iterations, 0 for unlimited

Default value: 0

`-M, --targetThroughput {targetThroughput}`

Target average throughput to achieve

Default value: 0

`-s, --searchScope {searchScope}`

Search scope ('base', 'one', 'sub', or 'subordinate')

Default value: sub

`-S, --scriptFriendly`

Use script-friendly mode

LDAP Connection Options

`-D, --bindDN {bindDN}`

DN to use to bind to the server

Default value: cn=Directory Manager

`-E, --reportAuthzID`

Use the authorization identity control

`-h, --hostname {host}`

Directory server hostname or IP address

Default value: localhost.localdomain

`-j, --bindPasswordFile {bindPasswordFile}`

Bind password file

`-K, --keyStorePath {keyStorePath}`

Certificate key store path

`-N, --certNickname {nickname}`

Nickname of certificate for SSL client authentication

`-o, --saslOption {name=value}`

SASL bind options

`-p, --port {port}`

Directory server port number

Default value: 389

`-P, --trustStorePath {trustStorePath}`

Certificate trust store path

`-q, --useStartTLS`

Use StartTLS to secure communication with the server

-T, --trustStorePassword {trustStorePassword}

Certificate trust store PIN

-u, --keyStorePasswordFile {keyStorePasswordFile}

Certificate key store PIN file

-U, --trustStorePasswordFile {path}

Certificate trust store PIN file

--usePasswordPolicyControl

Use the password policy request control

-w, --bindPassword {bindPassword}

Password to use to bind to the server

-W, --keyStorePassword {keyStorePassword}

Certificate key store PIN

-X, --trustAll

Trust all server SSL certificates

-Z, --useSSL

Use SSL for secure communication with the server

Utility Input/Output Options

--noPropertiesFile

No properties file will be used to get default command line argument values

--propertiesFilePath {propertiesFilePath}

Path to the file containing default property values used for command line arguments

-v, --verbose

Use verbose mode

General Options

-V, --version

Display version information

-?, -H, --help

Display usage information

Exit Codes

0

The command completed successfully.

89

An error occurred while parsing the command-line arguments.

Examples

The following example demonstrates measuring simple bind performance.

```
$ authrate -p 1389 -g "rand(names.txt)"
 -D "uid=%s,ou=people,dc=example,dc=com" -w password -c 10 -f
-----------------------------------------------------------------
     Throughput                            Response Time
   (ops/second)                           (milliseconds)
recent  average  recent  average  99.9%  99.99%  99.999%  err/sec
-----------------------------------------------------------------
9796.9   9816.6   1.029    1.029  12.413  161.451  161.835      0.0
14201.1  12028.1   0.704    0.835  9.508  161.456  167.573      0.0
14450.0  12835.9   0.692    0.782  8.989  161.835  174.518      0.0
12934.3  12860.6   0.773    0.779  9.253  161.339  174.426      0.0
14154.5  13121.0   0.706    0.764  9.025  161.451  177.101      0.0
^C
```

The `names.txt` contains all the user IDs for the sample suffix, and all user password values have been set to `password` for this example.

Name

ldapcompare — perform LDAP compare operations

Synopsis

`ldapcompare` {options} [[attribute] | [:] | [value]] [DN...]

Description

This utility can be used to perform LDAP compare operations in the directory.

Options

The following options are supported.

`--assertionFilter {filter}`

Use the LDAP assertion control with the provided filter

`-c, --continueOnError`

Continue processing even if there are errors

`-f, --filename {file}`

LDIF file containing one DN per line of entries to compare

`-J, --control {controloid[:criticality[:value|::b64value|:<filePath]]}`

Use a request control with the provided information

`-n, --dry-run`

Show what would be done but do not perform any operation

`-Y, --proxyAs {authzID}`

Use the proxied authorization control with the given authorization ID

LDAP Connection Options

`-D, --bindDN {bindDN}`

DN to use to bind to the server

Default value: cn=Directory Manager

-E, --reportAuthzID

Use the authorization identity control

-h, --hostname {host}

Directory server hostname or IP address

Default value: localhost.localdomain

-j, --bindPasswordFile {bindPasswordFile}

Bind password file

-K, --keyStorePath {keyStorePath}

Certificate key store path

-N, --certNickname {nickname}

Nickname of certificate for SSL client authentication

-o, --saslOption {name=value}

SASL bind options

-p, --port {port}

Directory server port number

Default value: 389

-P, --trustStorePath {trustStorePath}

Certificate trust store path

-q, --useStartTLS

Use StartTLS to secure communication with the server

-T, --trustStorePassword {trustStorePassword}

Certificate trust store PIN

-u, --keyStorePasswordFile {keyStorePasswordFile}

Certificate key store PIN file

-U, --trustStorePasswordFile {path}

Certificate trust store PIN file

--usePasswordPolicyControl

Use the password policy request control

`-V, --ldapVersion {version}`

LDAP protocol version number

Default value: 3

`-w, --bindPassword {bindPassword}`

Password to use to bind to the server

`-W, --keyStorePassword {keyStorePassword}`

Certificate key store PIN

`-X, --trustAll`

Trust all server SSL certificates

`-Z, --useSSL`

Use SSL for secure communication with the server

Utility Input/Output Options

`-i, --encoding {encoding}`

Use the specified character set for command-line input

`--noPropertiesFile`

No properties file will be used to get default command line argument values

`--propertiesFilePath {propertiesFilePath}`

Path to the file containing default property values used for command line arguments

`-v, --verbose`

Use verbose mode

General Options

`--version`

Display version information

`-?, -H, --help`

Display usage information

Exit Codes

0

The command completed successfully.

ldap-error

An LDAP error occurred while processing the operation.

LDAP result codes are described in RFC 4511. Also see the additional information for details.

89

An error occurred while parsing the command-line arguments.

Files

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

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

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

Examples

The following examples demonstrate comparing Babs Jensen's UID.

The following example uses a matching UID value.

```
$ ldapcompare -p 1389 uid:bjensen uid=bjensen,ou=people,dc=example,dc=com
Comparing type uid with value bjensen in entry
uid=bjensen,ou=people,dc=example,dc=com
Compare operation returned true for entry
uid=bjensen,ou=people,dc=example,dc=com
```

The following example uses a UID value that does not match.

```
$ ldapcompare -p 1389 uid:beavis uid=bjensen,ou=people,dc=example,dc=com
Comparing type uid with value beavis in entry
uid=bjensen,ou=people,dc=example,dc=com
Compare operation returned false for entry
uid=bjensen,ou=people,dc=example,dc=com
```

Name

ldapmodify — perform LDAP modify, add, delete, mod DN operations

Synopsis

`ldapmodify` {options}

Description

This utility can be used to perform LDAP modify, add, delete, and modify DN operations in the directory.

When not using a file to specify modifications, end your input with EOF (Ctrl+D on UNIX, Ctrl+Z on Windows).

Options

The following options are supported.

`-a, --defaultAdd`

Treat records with no changetype as add operations

`--assertionFilter {filter}`

Use the LDAP assertion control with the provided filter

`-c, --continueOnError`

Continue processing even if there are errors

`-f, --filename {file}`

LDIF file containing the changes to apply

`-J, --control {controloid[:criticality[:value|::b64value|:<filePath]]}`

Use a request control with the provided information

`-n, --dry-run`

Show what would be done but do not perform any operation

`--postReadAttributes {attrList}`

Use the LDAP ReadEntry post-read control

`--preReadAttributes {attrList}`

Use the LDAP ReadEntry pre-read control

`-Y, --proxyAs {authzID}`

Use the proxied authorization control with the given authorization ID

LDAP Connection Options

`-D, --bindDN {bindDN}`

DN to use to bind to the server

Default value: cn=Directory Manager

`-E, --reportAuthzID`

Use the authorization identity control

`-h, --hostname {host}`

Directory server hostname or IP address

Default value: localhost.localdomain

`-j, --bindPasswordFile {bindPasswordFile}`

Bind password file

`-K, --keyStorePath {keyStorePath}`

Certificate key store path

`-N, --certNickname {nickname}`

Nickname of certificate for SSL client authentication

`-o, --saslOption {name=value}`

SASL bind options

`-p, --port {port}`

Directory server port number

Default value: 389

`-P, --trustStorePath {trustStorePath}`

Certificate trust store path

-q, --useStartTLS

Use StartTLS to secure communication with the server

-T, --trustStorePassword {trustStorePassword}

Certificate trust store PIN

-u, --keyStorePasswordFile {keyStorePasswordFile}

Certificate key store PIN file

-U, --trustStorePasswordFile {path}

Certificate trust store PIN file

--usePasswordPolicyControl

Use the password policy request control

-V, --ldapVersion {version}

LDAP protocol version number

Default value: 3

-w, --bindPassword {bindPassword}

Password to use to bind to the server

-W, --keyStorePassword {keyStorePassword}

Certificate key store PIN

-X, --trustAll

Trust all server SSL certificates

-Z, --useSSL

Use SSL for secure communication with the server

Utility Input/Output Options

-i, --encoding {encoding}

Use the specified character set for command-line input

--noPropertiesFile

No properties file will be used to get default command line argument values

--propertiesFilePath {propertiesFilePath}

Path to the file containing default property values used for command line arguments

-v, --verbose

Use verbose mode

General Options

--version

Display version information

-?, -H, --help

Display usage information

Exit Codes

0

The command completed successfully.

ldap-error

An LDAP error occurred while processing the operation.

LDAP result codes are described in RFC 4511. Also see the additional information for details.

89

An error occurred while parsing the command-line arguments.

Files

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

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

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

Examples

The following example demonstrates use of the command to add an entry to the directory.

```
$ cat newuser.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
uid: newuser
facsimileTelephoneNumber: +1 408 555 1213
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
givenName: New
cn: New User
cn: Real Name
telephoneNumber: +1 408 555 1212
sn: Jensen
roomNumber: 1234
homeDirectory: /home/newuser
uidNumber: 10389
mail: newuser@example.com
l: South Pole
ou: Product Development
ou: People
gidNumber: 10636

$ ldapmodify -p 1389 -a -f newuser.ldif
 -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
Processing ADD request for uid=newuser,ou=People,dc=example,dc=com
ADD operation successful for DN uid=newuser,ou=People,dc=example,dc=com
```

The following example demonstrates adding a Description attribute to the new user's entry.

```
$ cat newdesc.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
changetype: modify
add: description
description: A new user's entry

$ ldapmodify -p 1389 -f newdesc.ldif
 -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
Processing MODIFY request for uid=newuser,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=newuser,ou=People,dc=example,dc=com
```

The following example demonstrates changing the Description attribute for the new user's entry.

```
$ cat moddesc.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Another description

$ ldapmodify -p 1389 -f moddesc.ldif
 -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
Processing MODIFY request for uid=newuser,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=newuser,ou=People,dc=example,dc=com
```

The following example demonstrates deleting the new user's entry.

```
$ cat deluser.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
changetype: delete

$ ldapmodify -p 1389 -f deluser.ldif
 -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
Processing DELETE request for uid=newuser,ou=People,dc=example,dc=com
DELETE operation successful for DN uid=newuser,ou=People,dc=example,dc=com
```

Name

ldappasswordmodify — perform LDAP password modifications

Synopsis

`ldappasswordmodify` {options}

Description

This utility can be used to perform LDAP password modify operations in the directory.

Options

The following options are supported.

`-a, --authzID {authzID}`

Authorization ID for the user entry whose password should be changed

The authorization ID is a string having either the prefix `dn:` followed by the user's distinguished name, or the prefix `u:` followed by a user identifier that depends on the identity mapping used to match the user identifier to an entry in the directory. Examples include `dn:uid=bjensen,ou=People,dc=example,dc=com`, and, if we assume that `bjensen` is mapped to Barbara Jensen's entry, `u:bjensen`.

`-A, --provideDNForAuthzID`

Use the bind DN as the authorization ID for the password modify operation

`-c, --currentPassword {currentPassword}`

Current password for the target user

`-C, --currentPasswordFile {file}`

Path to a file containing the current password for the target user

`-F, --newPasswordFile {file}`

Path to a file containing the new password to provide for the target user

`-J, --control {controloid[:criticality[:value|::b64value|:<filePath]]}`

Use a request control with the provided information

`-n, --newPassword {newPassword}`

New password to provide for the target user

LDAP Connection Options

`-D, --bindDN {bindDN}`

DN to use to bind to the server

Default value: cn=Directory Manager

`-E, --reportAuthzID`

Use the authorization identity control

`-h, --hostname {host}`

Directory server hostname or IP address

Default value: localhost.localdomain

`-j, --bindPasswordFile {bindPasswordFile}`

Bind password file

`-K, --keyStorePath {keyStorePath}`

Certificate key store path

`-N, --certNickname {nickname}`

Nickname of certificate for SSL client authentication

`-o, --saslOption {name=value}`

SASL bind options

`-p, --port {port}`

Directory server port number

Default value: 389

`-P, --trustStorePath {trustStorePath}`

Certificate trust store path

`-q, --useStartTLS`

Use StartTLS to secure communication with the server

`-T, --trustStorePassword {trustStorePassword}`

Certificate trust store PIN

-u, --keyStorePasswordFile {keyStorePasswordFile}

Certificate key store PIN file

-U, --trustStorePasswordFile {path}

Certificate trust store PIN file

--usePasswordPolicyControl

Use the password policy request control

-V, --ldapVersion {version}

LDAP protocol version number

Default value: 3

-w, --bindPassword {bindPassword}

Password to use to bind to the server

-W, --keyStorePassword {keyStorePassword}

Certificate key store PIN

-X, --trustAll

Trust all server SSL certificates

-Z, --useSSL

Use SSL for secure communication with the server

Utility Input/Output Options

--noPropertiesFile

No properties file will be used to get default command line argument values

--propertiesFilePath {propertiesFilePath}

Path to the file containing default property values used for command line arguments

-?, -H, --help

Display usage information

-v, --verbose

Use verbose mode

General Options

--version

Display version information

-?, -H, --help

Display usage information

Exit Codes

0

The command completed successfully.

ldap-error

An LDAP error occurred while processing the operation.

LDAP result codes are described in RFC 4511. Also see the additional information for details.

89

An error occurred while parsing the command-line arguments.

Files

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

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

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

Examples

The following example demonstrates a user changing the password for her entry.

```
$ cat /tmp/currpwd.txt /tmp/newpwd.txt
bribery
secret12
$ ldappasswordmodify -p 1389 -C /tmp/currpwd.txt -N /tmp/newpwd.txt
-A -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
The LDAP password modify operation was successful
```

Name

ldapsearch — perform LDAP search operations

Synopsis

`ldapsearch` {options} [filter] [attributes...]

Description

This utility can be used to perform LDAP search operations in the directory.

Options

The following options are supported.

`-a, --dereferencePolicy {dereferencePolicy}`

Alias dereference policy ('never', 'always', 'search', or 'find')

Default value: never

`-A, --typesOnly`

Only retrieve attribute names but not their values

`--assertionFilter {filter}`

Use the LDAP assertion control with the provided filter

`-b, --baseDN {baseDN}`

Base DN format string

`-c, --continueOnError`

Continue processing even if there are errors

`-C, --persistentSearch ps[:changetype[:changesonly[:entrychgcontrols]]]`

Use the persistent search control

`--countEntries`

Count the number of entries returned by the server

`-e, --getEffectiveRightsAttribute {attribute}`

Specifies geteffectiverights control specific attribute list

-f, --filename {file}

LDIF file containing the changes to apply

-g, --getEffectiveRightsAuthzid {authzID}

Use geteffectiverights control with the provided authzid

-G, --virtualListView {before:after:index:count | before:after:value}

Use the virtual list view control to retrieve the specified results page

-J, --control {controloid[:criticality[:value|::b64value|:<filePath]]}

Use a request control with the provided information

-l, --timeLimit {timeLimit}

Maximum length of time in seconds to allow for the search

Default value: 0

--matchedValuesFilter {filter}

Use the LDAP matched values control with the provided filter

-n, --dry-run

Show what would be done but do not perform any operation

-s, --searchScope {searchScope}

Search scope ('base', 'one', 'sub', or 'subordinate')

Default value: sub

`subordinate` is an LDAP extension that might not work with all LDAP servers.

-S, --sortOrder {sortOrder}

Sort the results using the provided sort order

--simplePageSize {numEntries}

Use the simple paged results control with the given page size

Default value: 1000

-Y, --proxyAs {authzID}

Use the proxied authorization control with the given authorization ID

-z, --sizeLimit {sizeLimit}

Maximum number of entries to return from the search

Default value: 0

LDAP Connection Options

-D, --bindDN {bindDN}

DN to use to bind to the server

Default value: cn=Directory Manager

-E, --reportAuthzID

Use the authorization identity control

-h, --hostname {host}

Directory server hostname or IP address

Default value: localhost.localdomain

-j, --bindPasswordFile {bindPasswordFile}

Bind password file

-K, --keyStorePath {keyStorePath}

Certificate key store path

-N, --certNickname {nickname}

Nickname of certificate for SSL client authentication

-o, --saslOption {name=value}

SASL bind options

-p, --port {port}

Directory server port number

Default value: 389

-P, --trustStorePath {trustStorePath}

Certificate trust store path

-q, --useStartTLS

Use StartTLS to secure communication with the server

`-T, --trustStorePassword {trustStorePassword}`

Certificate trust store PIN

`-u, --keyStorePasswordFile {keyStorePasswordFile}`

Certificate key store PIN file

`-U, --trustStorePasswordFile {path}`

Certificate trust store PIN file

`--usePasswordPolicyControl`

Use the password policy request control

`-V, --ldapVersion {version}`

LDAP protocol version number

Default value: 3

`-w, --bindPassword {bindPassword}`

Password to use to bind to the server

`-W, --keyStorePassword {keyStorePassword}`

Certificate key store PIN

`-X, --trustAll`

Trust all server SSL certificates

`-Z, --useSSL`

Use SSL for secure communication with the server

Utility Input/Output Options

`-i, --encoding {encoding}`

Use the specified character set for command-line input

`--noPropertiesFile`

No properties file will be used to get default command line argument values

`--propertiesFilePath {propertiesFilePath}`

Path to the file containing default property values used for command line arguments

`-t, --dontWrap`

Do not wrap long lines

`-v, --verbose`

Use verbose mode

General Options

`--version`

Display version information

`-?, -H, --help`

Display usage information

Filter

The filter argument is a string representation of an LDAP search filter as in `(cn=Babs Jensen)`, `(&(objectClass=Person)(|(sn=Jensen)(cn=Babs J*)))`, or `(cn:caseExactMatch:=Fred Flintstone)`.

Attribute

The optional attribute list specifies the attributes to return in the entries found by the search. In addition to identifying attributes by name such as `cn sn mail` and so forth, you can use the following notations, too.

`*`

Return all user attributes such as `cn`, `sn`, and `mail`.

`+`

Return all operational attributes such as `etag` and `pwdPolicySubentry`.

`@objectclass`

Return all attributes of the specified object class, where *objectclass* is one of the object classes on the entries returned by the search.

Exit Codes

0

The command completed successfully.

ldap-error

An LDAP error occurred while processing the operation.

LDAP result codes are described in RFC 4511. Also see the additional information for details.

89

An error occurred while parsing the command-line arguments.

Files

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

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

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

Examples

The following example searches for entries with UID containing `jensen`, returning only DNs and uid values.

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

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

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

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

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

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

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


Result Code:  0 (Success)
```

You can also use `@objectclass` notation in the attribute list to return the attributes of a particular object class. The following example shows how to return attributes of the `inetOrgPerson` object class.

```
$ ldapsearch -p 1389 -b dc=example,dc=com "(uid=bjensen)" @inetorgperson
dn: uid=bjensen,ou=People,dc=example,dc=com
givenName: Barbara
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
uid: bjensen
cn: Barbara Jensen
cn: Babs Jensen
telephoneNumber: +1 408 555 1862
sn: Jensen
roomNumber: 0209
mail: bjensen@example.com
l: Cupertino
ou: Product Development
ou: People
facsimileTelephoneNumber: +1 408 555 1992
```

You can use `+` in the attribute list to return all operational attributes, as in the following example.

```
$ ldapsearch -p 1389 -b dc=example,dc=com "(uid=bjensen)" +
dn: uid=bjensen,ou=People,dc=example,dc=com
numSubordinates: 0
structuralObjectClass: inetOrgPerson
etag: 0000000073c29972
pwdPolicySubentry: cn=Default Password Policy,cn=Password Policies,cn=config
subschemaSubentry: cn=schema
hasSubordinates: false
entryDN: uid=bjensen,ou=people,dc=example,dc=com
entryUUID: fc252fd9-b982-3ed6-b42a-c76d2546312c
```

Name

ldifdiff — compare small LDIF files

Synopsis

`ldifdiff` [options] *source*.ldif *target*.ldif

Description

This utility can be used to compare two LDIF files and report the differences in LDIF format.

Options

The following options are supported.

`-a, --ignoreAttrs {file}`

File containing a list of attributes to ignore when computing the difference.

`--checkSchema`

Takes into account the syntax of the attributes as defined in the schema to make the value comparison. The provided LDIF files must conform to the server schema.

`-e, --ignoreEntries {file}`

File containing a list of entries (DN) to ignore when computing the difference.

`-S, --singleValueChanges`

Each attribute-level change should be written as a separate modification per attribute value rather than one modification per entry.

`-V, --version`

Display version information.

`-?, -H, --help`

Display usage information.

Exit Codes

0

The command completed successfully.

> 0

An error occurred.

Examples

The following example demonstrates use of the command with two small LDIF files.

```
$ cat /path/to/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

$ cat /path/to/neweruser.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: secret12
description: A new description.

$ ldifdiff /path/to/newuser.ldif /path/to/neweruser.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
changetype: modify
add: userPassword
userPassword: secret12
-
delete: userPassword
userPassword: changeme
-
add: description
description: A new description.
```

Name

ldifmodify — apply LDIF changes to LDIF

Synopsis

`ldifmodify` {options} source [changes]

Description

This utility can be used to apply a set of modify, add, and delete operations against data in an LDIF file.

Options

The following options are supported.

`-c, --continueOnError`

Continue processing even if there are errors

`-o, --outputLDIF {file}`

Write updated entries to {file} instead of stdout

Default value: stdout

`-V, --version`

Display version information.

`-?, -H, --help`

Display usage information.

Exit Codes

0

The command completed successfully.

> 0

An error occurred.

Examples

The following example demonstrates use of the command.

```
$ cat /path/to/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

$ cat /path/to/newdiff.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
changetype: modify
add: userPassword
userPassword: secret12
-
delete: userPassword
userPassword: changeme
-
add: description
description: A new description.

$ ldifmodify -o neweruser.ldif /path/to/newuser.ldif /path/to/newdiff.ldif
$ cat neweruser.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: secret12
description: A new description.
```

Name

ldifsearch — search LDIF with LDAP filters

Synopsis

`ldifsearch` {options} source [filter] [attribute...]

Description

This utility can be used to perform search operations against data in an LDIF file.

Options

The following options are supported.

`-A, --typesOnly`

Only retrieve attribute names but not their values

`-b, --baseDN {baseDN}`

Search base DN

`-f, --filterFile {filterFile}`

File containing a list of search filter strings

`-l, --timeLimit {timeLimit}`

Maximum length of time in seconds to allow for the search

Default value: 0

`-o, --outputFile {File}`

Write search results to {file} instead of stdout

Default: stdout

`-s, --searchScope {scope}`

Search scope ('base', 'one', 'sub', or 'subordinate')

Default value: sub

`-V, --version`

Display version information.

`-z, --sizeLimit {sizeLimit}`

Maximum number of matching entries to return from the search

Default value: 0

`-?, -H, --help`

Display usage information.

Filter

The filter argument is a string representation of an LDAP search filter as in `(cn=Babs Jensen)`, `(&(objectClass=Person)(|(sn=Jensen)(cn=Babs J*)))`, or `(cn:caseExactMatch:=Fred Flintstone)`.

Attribute

The optional attribute list specifies the attributes to return in the entries found by the search. In addition to identifying attributes by name such as `cn sn mail` and so forth, you can use the following notations, too.

`*`

Return all user attributes such as `cn`, `sn`, and `mail`.

`+`

Return all operational attributes such as `etag` and `pwdPolicySubentry`.

`@objectclass`

Return all attributes of the specified object class, where *objectclass* is one of the object classes on the entries returned by the search.

Exit Codes

0

The command completed successfully.

> 0

An error occurred.

Examples

The following example demonstrates use of the command.

```
$ ldifsearch -b dc=example,dc=com /path/to/Example.ldif uid=bjensen
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
uid: bjensen
userpassword: hifalutin
facsimiletelephonenumber: +1 408 555 1992
givenname: Barbara
cn: Barbara Jensen
cn: Babs Jensen
telephonenumber: +1 408 555 1862
sn: Jensen
roomnumber: 0209
homeDirectory: /home/bjensen
mail: bjensen@example.com
l: Cupertino
ou: Product Development
ou: People
uidNumber: 1076
gidNumber: 1000
```

You can also use *`@objectclass`* notation in the attribute list to return the attributes of a particular object class. The following example shows how to return attributes of the `posixAccount` object class.

```
$ ldifsearch --ldifFile /path/to/Example.ldif
 --baseDN dc=example,dc=com "(uid=bjensen)" @posixaccount
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
uid: bjensen
userpassword: hifalutin
cn: Barbara Jensen
cn: Babs Jensen
homeDirectory: /home/bjensen
uidNumber: 1076
gidNumber: 1000
```

Name

modrate — measure modification throughput and response time

Synopsis

`modrate` {options} [[attribute] | [:] | [value format string]...]

Description

This utility can be used to measure modify throughput and response time of a directory service using user-defined modifications.

Options

The following options are supported.

`-A, --asynchronous`

Use asynchronous mode and don't wait for results before sending the next request

`-b, --baseDN {baseDN}`

Base DN format string

`-c, --numConnections {numConnections}`

Number of connections

Default value: 1

`-e, --percentile {percentile}`

Calculate max response time for a percentile of operations

`-f, --keepConnectionsOpen`

Keep connections open

`-F, --noRebind`

Keep connections open and don't rebind

`-g, --argument {generator function or static string}`

Argument used to evaluate the Java style format strings in program parameters (Base DN, Search Filter). The set of all arguments provided form the the argument list in order. Besides static string arguments, they can be generated per iteration with the following functions:

"inc({filename})"

Consecutive, incremental line from file

"inc({min},{max})"

Consecutive, incremental number

"rand({filename})"

Random line from file

"rand({min},{max})"

Random number

"randstr({length},*charSet*)"

Random string of specified length and optionally from characters in the charSet string. A range of character can be specified with [start-end] charSet notation. If no charSet is specified, the default charSet of [A-Z][a-z][0-9] will be used.

`-i, --statInterval {statInterval}`

Display results each specified number of seconds

Default value: 5

`-m, --maxIterations {maxIterations}`

Max iterations, 0 for unlimited

Default value: 0

`-M, --targetThroughput {targetThroughput}`

Target average throughput to achieve

Default value: 0

`-S, --scriptFriendly`

Use script-friendly mode

`-t, --numConcurrentTasks {numConcurrentTasks}`

Number of concurrent tasks per connection

Default value: 1

LDAP Connection Options

`-D, --bindDN {bindDN}`

DN to use to bind to the server

Default value: cn=Directory Manager

`-E, --reportAuthzID`

Use the authorization identity control

`-h, --hostname {host}`

Directory server hostname or IP address

Default value: localhost.localdomain

`-j, --bindPasswordFile {bindPasswordFile}`

Bind password file

`-K, --keyStorePath {keyStorePath}`

Certificate key store path

`-N, --certNickname {nickname}`

Nickname of certificate for SSL client authentication

`-o, --saslOption {name=value}`

SASL bind options

`-p, --port {port}`

Directory server port number

Default value: 389

`-P, --trustStorePath {trustStorePath}`

Certificate trust store path

`-q, --useStartTLS`

Use StartTLS to secure communication with the server

`-T, --trustStorePassword {trustStorePassword}`

Certificate trust store PIN

-u, --keyStorePasswordFile {keyStorePasswordFile}

Certificate key store PIN file

-U, --trustStorePasswordFile {path}

Certificate trust store PIN file

--usePasswordPolicyControl

Use the password policy request control

-w, --bindPassword {bindPassword}

Password to use to bind to the server

-W, --keyStorePassword {keyStorePassword}

Certificate key store PIN

-X, --trustAll

Trust all server SSL certificates

-Z, --useSSL

Use SSL for secure communication with the server

Utility Input/Output Options

--noPropertiesFile

No properties file will be used to get default command line argument values

--propertiesFilePath {propertiesFilePath}

Path to the file containing default property values used for command line arguments

-v, --verbose

Use verbose mode

General Options

-V, --version

Display version information

-?, -H, --help

Display usage information

Exit Codes

0

The command completed successfully.

89

An error occurred while parsing the command-line arguments.

Examples

The following example demonstrates testing directory performance by using the modrate command to write random 16-character description values to all entries in a sample file.

```
$ grep ^uid: /path/to/Example.ldif | sed -e "s/uid: //" > names.txt
$ modrate -p 1389 -D "cn=Directory Manager" -w password  -A -F -c 4 -t 4
 -b "uid=%s,ou=people,dc=example,dc=com" -g "rand(names.txt)"
 -g "randstr(16)" 'description:%2$s'
--------------------------------------------------------------------------
     Throughput                            Response Time
   (ops/second)                           (milliseconds)
recent  average  recent  average  99.9%  99.99%  99.999%  err/sec  req/res
--------------------------------------------------------------------------
1085.9   1088.5  993.849  993.849  2135.220  2510.361  2510.361  0.0  2.3
2086.7   1648.8  1963.980  1683.038  3015.025  3078.628  3215.050  0.0  1.0
3097.3   2092.6  1332.930  1524.278  2940.131  3024.811  3215.050  0.0  1.0
3848.3   2501.4  1045.000  1352.583  2902.235  3015.863  3215.050  0.0  1.0
3641.2   2717.4  1106.157  1290.003  2901.379  3015.597  3215.050  0.0  1.0
3759.4   2883.0  1065.732  1243.534  2900.400  3015.501  3215.050  0.0  1.0
^C
```

Name

searchrate — measure search throughput and response time

Synopsis

`searchrate` {options} [filter format string] [attributes...]

Description

This utility can be used to measure search throughput and response time of a directory service using user-defined searches.

Options

The following options are supported.

`-a, --dereferencePolicy {dereferencePolicy}`

Alias dereference policy ('never', 'always', 'search', or 'find')

Default value: never

`-A, --asynchronous`

Use asynchronous mode and don't wait for results before sending the next request

`-b, --baseDN {baseDN}`

Base DN format string

`-c, --numConnections {numConnections}`

Number of connections

Default value: 1

`-e, --percentile {percentile}`

Calculate max response time for a percentile of operations

`-f, --keepConnectionsOpen`

Keep connections open

`-F, --noRebind`

Keep connections open and don't rebind

`-g, --argument {generator function or static string}`

Argument used to evaluate the Java style format strings in program parameters (Base DN, Search Filter). The set of all arguments provided form the the argument list in order. Besides static string arguments, they can be generated per iteration with the following functions:

"inc({filename})"

Consecutive, incremental line from file

"inc({min},{max})"

Consecutive, incremental number

"rand({filename})"

Random line from file

"rand({min},{max})"

Random number

"randstr({length},*charSet*)"

Random string of specified length and optionally from characters in the charSet string. A range of character can be specified with [start-end] charSet notation. If no charSet is specified, the default charSet of [A-Z][a-z][0-9] will be used.

`-i, --statInterval {statInterval}`

Display results each specified number of seconds

Default value: 5

`-m, --maxIterations {maxIterations}`

Max iterations, 0 for unlimited

Default value: 0

`-M, --targetThroughput {targetThroughput}`

Target average throughput to achieve

Default value: 0

`-s, --searchScope {searchScope}`

Search scope ('base', 'one', 'sub', or 'subordinate')

Default value: sub

`-S, --scriptFriendly`

Use script-friendly mode

`-t, --numConcurrentTasks {numConcurrentTasks}`

Number of concurrent tasks per connection

Default value: 1

LDAP Connection Options

`-D, --bindDN {bindDN}`

DN to use to bind to the server

Default value: cn=Directory Manager

`-E, --reportAuthzID`

Use the authorization identity control

`-h, --hostname {host}`

Directory server hostname or IP address

Default value: localhost.localdomain

`-j, --bindPasswordFile {bindPasswordFile}`

Bind password file

`-K, --keyStorePath {keyStorePath}`

Certificate key store path

`-N, --certNickname {nickname}`

Nickname of certificate for SSL client authentication

`-o, --saslOption {name=value}`

SASL bind options

`-p, --port {port}`

Directory server port number

Default value: 389

`-P, --trustStorePath {trustStorePath}`

Certificate trust store path

`-q, --useStartTLS`

Use StartTLS to secure communication with the server

`-T, --trustStorePassword {trustStorePassword}`

Certificate trust store PIN

`-u, --keyStorePasswordFile {keyStorePasswordFile}`

Certificate key store PIN file

`-U, --trustStorePasswordFile {path}`

Certificate trust store PIN file

`--usePasswordPolicyControl`

Use the password policy request control

`-w, --bindPassword {bindPassword}`

Password to use to bind to the server

`-W, --keyStorePassword {keyStorePassword}`

Certificate key store PIN

`-X, --trustAll`

Trust all server SSL certificates

`-Z, --useSSL`

Use SSL for secure communication with the server

Utility Input/Output Options

`--noPropertiesFile`

No properties file will be used to get default command line argument values

`--propertiesFilePath {propertiesFilePath}`

Path to the file containing default property values used for command line arguments

-v, --verbose

Use verbose mode

General Options

-V, --version

Display version information

-?, -H, --help

Display usage information

Exit Codes

0

The command completed successfully.

89

An error occurred while parsing the command-line arguments.

Examples

The following example demonstrates measuring search performance.

```
$ grep ^uid: /path/to/Example.ldif | sed -e "s/uid: //" > names.txt
$ searchrate -p 1389 -b dc=example,dc=com -A -F -c 4 -t 4
 -g "rand(names.txt)" "(uid=%s)"
-------------------------------------------------------------------------------
     Throughput                            Response Time
   (ops/second)                           (milliseconds)
recent  average  recent  average  99.9%  99.99%  99.999%  err/sec  Entries/Srch
-------------------------------------------------------------------------------
1475.9   1475.9   0.423    0.423  6.938  126.236 126.236      0.0           1.0
2596.5   2038.4   0.254    0.315  6.866  12.980  126.236      0.0           1.0
3210.7   2428.2   0.205    0.267  5.733  11.710  126.236      0.0           1.0
3080.5   2591.0   0.215    0.252  5.733  10.541  126.236      0.0           1.0
3236.9   2720.1   0.203    0.240  5.258  10.514  126.236      0.0           1.0
3181.1   2796.8   0.207    0.234  5.258  10.384  126.236      0.0           1.0
3202.5   2854.8   0.206    0.229  4.825  10.384  126.236      0.0           1.0
^C
```

OpenDJ Glossary

Abandon operation	LDAP operation to stop processing of a request in progress, after which the directory server drops the connection without a reply to the client application.
Access control	Control to grant or to deny access to a resource.
Access control instruction (ACI)	Instruction added as a directory entry attribute for fine-grained control over what a given user or group member is authorized to do in terms of LDAP operations and access to user data. ACIs are implemented independently from privileges, which apply to administrative operations. See also Privilege.
Access control list (ACL)	An access control list connects a user or group of users to one or more security entitlements. For example, users in group "sales" are granted the entitlement "read-only" to some financial data.
`access` log	Directory server log tracing the operations the server processes including timestamps, connection information, and information about the operation itself.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Active user	A user that has the ability to authenticate and use the services, having valid credentials.
Add operation	LDAP operation to add a new entry or entries to the directory.

Anonymous	A user that does not need to authenticate, and is unknown to the system.
Anonymous bind	A bind operation using simple authentication with an empty DN and an empty password, allowing "anonymous" access such as reading public information.
Approximate index	Index is used to match values that "sound like" those provided in the filter.
Attribute	Properties of a directory entry, stored as one or more key-value pairs. Typical examples include the common name (`cn`) to store the user's full name and variations of the name, user ID (`uid`) to store a unique identifier for the entry, and `mail` to store email addresses.
`audit` log	Type of access log that dumps changes in LDIF.
Authentication	The process of verifying who is requesting access to a resource; the act of confirming the identity of a principal.
Authorization	The process of determining whether access should be granted to an individual based on information about that individual; the act of determining whether to grant or to deny a principal access to a resource.
Backend	Repository that a directory server can access to store data. Different implementations with different capabilities exist.
Binary copy	Binary backup archive of one directory server that can be restored on another directory server.
Bind operation	LDAP authentication operation to determine the client's identity in LDAP terms, the identity which is later used by the server to authorize (or not) access to directory data that the client wants to lookup or change.
Collective attribute	A standard mechanism for defining attributes that appear on all the entries in a particular subtree.
Compare operation	LDAP operation to compare a specified attribute value with the value stored on an entry in the directory.
Control	Information added to an LDAP message to further specify how an LDAP operation should be processed. OpenDJ supports many LDAP controls.
Database cache	Memory space set aside to hold database content.
`debug` log	Directory server log tracing details needed to troubleshoot a problem in the server.

Delete operation	LDAP operation to remove an existing entry or entries from the directory.
Directory	A directory is a network service which lists participants in the network such as users, computers, printers, and groups. The directory provides a convenient, centralized, and robust mechanism for publishing and consuming information about network participants.
Directory hierarchy	A directory can be organized into a hierarchy in order to make it easier to browse or manage. Directory hierarchies normally represent something in the physical world, such as organizational hierarchies or physical locations. For example, the top level of a directory may represent a company, the next level down divisions, the next level down departments, and so on. Alternately, the top level may represent the world, the next level down countries, next states or provinces, next cities, and so on.
Directory manager	Default Root DN who has privileges to do full administration of the OpenDJ server, including bypassing access control evaluation, changing access controls, and changing administrative privileges. See also Root DN.
Directory object	A directory object is an item in a directory. Example objects include users, user groups, computers and more. Objects may be organized into a hierarchy and contain identifying attributes. See also Entry.
Directory server	Server application for centralizing information about network participants. A highly available directory service consists of multiple directory servers configured to replicate directory data. See also Directory, Replication.
Directory Services Markup Language (DSML)	Standard language to access directory services using XML. DMSL v1 defined an XML mapping of LDAP objects, while DSMLv2 maps the LDAP Protocol and data model to XML.
Distinguished name (DN)	Fully qualified name for a directory entry, such as `uid=bjensen,ou=People,dc=example,dc=com`, built by concatenating the entry RDN (`uid=bjensen`) with the DN of the parent entry (`ou=People,dc=example,dc=com`).
Dynamic group	Group that specifies members using LDAP URLs.
Entry	As generic and hierarchical data stores, directories always contain different kinds of entries, either nodes (or containers) or leaf entries. An entry is an object in the directory, defined by one of more object classes and their related attributes. At startup, OpenDJ reports the number of entries contained in each suffix.
Entry cache	Memory space set aside to hold frequently-accessed, large entries, such as static groups.

Equality index	Index used to match values that correspond exactly (though generally without case sensitivity) to the value provided in the search filter.
`errors` log	Directory server log tracing server events, error conditions, and warnings, categorized and identified by severity.
Export	Save directory data in an LDIF file.
Extended operation	Additional LDAP operation not included in the original standards. OpenDJ supports several standard LDAP extended operations.
Extensible match index	Index for a matching rule other than approximate, equality, ordering, presence, substring or VLV, such as an index for generalized time.
External user	An individual that accesses company resources or services but is not working for the company. Typically a customer or partner.
Filter	An LDAP search filter is an expression that the server uses to find entries that match a search request, such as `(mail=*@example.com)` to match all entries having an email address in the example.com domain.
Group	Entry identifying a set of members whose entries are also in the directory.
Idle time limit	Defines how long OpenDJ allows idle connections to remain open.
Import	Read in and index directory data from an LDIF file.
Inactive user	An entry in the directory that once represented a user but which is now no longer able to be authenticated.
Index	Directory server backend feature to allow quick lookup of entries based on their attribute values. See also Approximate index, Equality index, Extensible match index, Ordering index, Presence index, Substring index, Virtual list view (VLV) index, Index entry limit.
Index entry limit	When the number of entries that an index key points to exceeds the index entry limit, OpenDJ stops maintaining the list of entries for that index key.
Internal user	An individual who works within the company either as an employee or as a contractor.
LDAP Data Interchange Format (LDIF)	Standard, portable, text-based representation of directory content. See RFC 2849.
LDAP URL	LDAP Uniform Resource Locator such as `ldap://directory.example.com:389/dc=example,dc=com??sub?(uid=bjensen)`. See RFC 2255.

LDAPS	LDAP over SSL.
Lightweight Directory Access Protocol (LDAP)	A simple and standardized network protocol used by applications to connect to a directory, search for objects and add, edit or remove objects. See RFC 4510.
Lookthrough limit	Defines the maximum number of candidate entries OpenDJ considers when processing a search.
Matching rule	Defines rules for performing matching operations against assertion values. Matching rules are frequently associated with an attribute syntax and are used to compare values according to that syntax. For example, the `distinguishedNameEqualityMatch` matching rule can be used to determine whether two DNs are equal and can ignore unnecessary spaces around commas and equal signs, differences in capitalization in attribute names, and so on.
Modify DN operation	LDAP modification operation to request that the server change the distinguished name of an entry.
Modify operation	LDAP modification operation to request that the server change one or more attributes of an entry.
Naming context	Base DN under which client applications can look for user data.
Object class	Identifies entries that share certain characteristics. Most commonly, an entry's object classes define the attributes that must and may be present on the entry. Object classes are stored on entries as values of the `objectClass` attribute. Object classes are defined in the directory schema, and can be abstract (defining characteristics for other object classes to inherit), structural (defining the basic structure of an entry, one structural inheritance per entry), or auxiliary (for decorating entries already having a structural object class with other required and optional attributes).
Object identifier (OID)	String that uniquely identifies an object, such as `0.9.2342.19200300.100.1.1` for the user ID attribute or `1.3.6.1.4.1.1466.115.121.1.15` for `DirectoryString` syntax.
Operational attribute	An attribute that has a special (operational) meaning for the directory server, such as `pwdPolicySubentry` or `modifyTimestamp`.
Ordering index	Index used to match values for a filter that specifies a range.
Password policy	A set of rules regarding what sequence of characters constitutes an acceptable password. Acceptable passwords are generally those that would be too difficult for another user or an automated program to guess and thereby defeat the password mechanism. Password policies may require a minimum length, a mixture of different types

	of characters (lowercase, uppercase, digits, punctuation marks, and so forth), avoiding dictionary words or passwords based on the user's name, and so forth. Password policies may also require that users not reuse old passwords and that users change their passwords regularly.
Password reset	Password change performed by a user other than the user who owns the entry.
Password storage scheme	Mechanism for encoding user passwords stored on directory entries. OpenDJ implements a number of password storage schemes.
Password validator	Mechanism for determining whether a proposed password is acceptable for use. OpenDJ implements a number of password validators.
Presence index	Index used to match the fact that an attribute is present on the entry, regardless of the value.
Principal	Entity that can be authenticated, such as a user, a device, or an application.
Privilege	Server configuration settings controlling access to administrative operations such as exporting and importing data, restarting the server, performing password reset, and changing the server configuration. Privileges are implemented independently from access control instructions (ACI), which apply to LDAP operations and user data. See also Access control instruction (ACI).
Referential integrity	Ensuring that group membership remains consistent following changes to member entries.
`referint` log	Directory server log tracing referential integrity events, with entries similar to the errors log.
Referral	Reference to another directory location, which can be another directory server running elsewhere or another container on the same server, where the current operation can be processed.
Relative distinguished name (RDN)	Initial portion of a DN that distinguishes the entry from all other entries at the same level, such as `uid=bjensen` in `uid=bjensen,ou=People,dc=example,dc=com`.
Replication	Data synchronization that ensures all directory servers participating eventually share a consistent set of directory data.
`replication` log	Directory server log tracing replication events, with entries similar to the errors log.

Root DN	A directory superuser, whose account is specific to a directory server under `cn=Root DNs,cn=config`. The default Root DN is Directory Manager. You can create additional Root DN accounts, each with different administrative privileges. See also Directory manager, Privilege.
Root DSE	The directory entry with distinguished name "" (empty string), where DSE stands for DSA-Specific Entry. DSA stands for Directory Server Agent, a single directory server. The root DSE serves to expose information over LDAP about what the directory server supports in terms of LDAP controls, auth password schemes, SASL mechanisms, LDAP protocol versions, naming contexts, features, LDAP extended operations, and so forth.
Schema	LDAP schema defines the object classes, attributes types, attribute value syntaxes, matching rules and so on that constrain entries held by the directory server.
Search filter	See Filter.
Search operation	LDAP lookup operation where a client requests that the server return entries based on an LDAP filter and a base DN under which to search.
Simple authentication	Bind operation performed with a user's entry DN and user's password. Use simple authentication only if the network connection is secure.
Size limit	Sets the maximum number of entries returned for a search.
Static group	Group that enumerates member entries.
Subentry	An entry, such as a password policy entry, that resides with the user data but holds operational data, and is not visible in search results unless explicitly requested.
Substring index	Index used to match values specified with wildcards in the filter.
Task	Mechanism to provide remote access to directory server administrative functions. OpenDJ supports tasks to backup and restore backends, to import and export LDIF files, and to stop and restart the server.
Time limit	Defines the maximum processing time OpenDJ devotes to a search operation.
Unbind operation	LDAP operation to release resources at the end of a session.
Unindexed search	Search operation for which no matching index is available. If no indexes are applicable, then the directory server potentially has to go through all entries to look for candidate matches. For this reason, the

	`unindexed-search` privilege, allowing users to request searches for which no applicable index exists, is reserved for the directory manager by default.
User	An entry that represents an individual that can be authenticated through credentials contained or referenced by its attributes. A user may represent an internal user or an external user, and may be an active user or an inactive user.
User attribute	An attribute for storing user data on a directory entry such as `mail` or `givenname`.
Virtual attribute	An attribute with dynamically generated values that appear in entries but are not persistently stored in the backend.
Virtual directory	An application that exposes a consolidated view of multiple physical directories over an LDAP interface. Consumers of the directory information connect to the virtual directory's LDAP service. Behind the scenes, requests for information and updates to the directory are sent to one or more physical directories where the actual information resides. Virtual directories enable organizations to create a consolidated view of information that for legal or technical reasons cannot be consolidated into a single physical copy.
Virtual list view (VLV) index	Browsing index designed to help the directory server respond to client applications that need for example to browse through a long list of results a page at a time in a GUI.
Virtual static group	OpenDJ group that lets applications see dynamic groups as what appear to be static groups.
X.500	A family of standardized protocols for accessing, browsing and maintaining a directory. X.500 is functionally similar to LDAP, but is generally considered to be more complex, and has consequently not been widely adopted.

Index