Netscape Directory SDK 4.0 for Java Programmer's Guide

Netscape Directory SDK for Java

Version 4.0


Recycled and Recyclable Paper

The Team:

Engineering:	Rob Weltman, Miodrag Kekic
Marketing:	Michael Mullany
Publications:	Gina Cariaga, Jacob Rosenschein
Quality Assurance:	Sudesh Chandra, Tobias Crawley
Release:	Walt Miller
Support:	Thanos Foufoulas

Version 4.0

Printed in USA
00 99 98 10 9 8 7 6 5 4 3 2 1

Netscape Communications Corporation 501 East Middlefield Road, Mountain View, CA 94043

Contents

Preface

The *Netscape Directory SDK 4.0 for Java Programmer's Guide* documents the Netscape Directory SDK 4.0 for Java, a development kit for writing LDAP (Lightweight Directory Access Protocol) applications.

The chapter has the following sections:

- "Who Should Read This Guide"
- "Where to Find LDAP Information"
- "What's New in This Release"
- "What's in This Guide"
- "Where to Find Reference Information"
- "Documentation Conventions"
- "About the Sample Code"

Who Should Read This Guide

This guide is intended for use by Java programmers who want to enable new or existing applications and applets to connect to, search, and update LDAP servers. It assumes that you are familiar with writing and compiling Java applications and applets. If you plan to use the LDAP Java Beans, you should also be familiar with Java Beans.

Where to Find LDAP Information

To find more information about the LDAP protocol and directories, see LDAP and Directory Developer Central at the Netscape DevEdge site:

`http://developer.netscape.com/tech/directory/index.html`

What's New in This Release

This manual accompanies the Netscape Directory SDK 4.0 for Java release. This SDK supports the LDAP v3 protocol and includes classes and methods for:

- working with LDAP v3 controls (for example, to request server-side sorting of search results)
- requesting LDAP v3 extended operations and parsing extended responses
- handling search references
- using a SASL mechanism for authentication
- interacting with the Java Naming and Directory Interface (JNDI)
- managing asynchronous connections

The Netscape Directory SDK 4.0 for Java supports the specifications detailed in the Internet-Draft "The Java LDAP Application Program Interface". This Internet-Draft is available at `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-java-api-06.txt`

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

For late-breaking information about the Netscape Directory SDK 4.0 for Java, read the Release Notes at:

`http://developer.netscape.com/docs/manuals/dirsdk/jsdk40/relnotes.html`

What's in This Guide

This guide explains how to use the Netscape Directory SDK 4.0 for Java to enable applications to interact with LDAP servers. The guide documents the Java LDAP API, which consists of interfaces, classes, and methods used to communicate with LDAP servers.

This manual is organized into three parts:

- Part 1, “Introduction to LDAP and the Netscape Directory SDK for Java”, explains how you can use the Netscape Directory SDK 4.0 for Java to enable your applications for LDAP.
- Part 2, “Writing Clients with the Netscape Directory SDK for Java” teaches you how to employ the LDAP Java API in your software.
- Part 3, “Advanced Topics”, contains additional material that you might need, including documentation of the classes and methods that support LDAP v3 features, a discussion of secure authentication methods and information about JNDI and asynchronous communication.

Where to Find Reference Information

All interface, class and method reference materials for the Netscape Directory SDK 4.0 for Java are generated using the Javadoc utility. You can access this information either online at: `http://developer.netscape.com/docs/manuals/dirsdk/jsdk40/Reference/` or by pointing your Web browser to the `index.html` file in the `/dist/doc` subdirectory of your SDK installation. Any updates or changes to these materials are reflected in the online references.

Documentation Conventions

This book uses the following font conventions:

- `The monospace font` is used for sample code and code listings, API and language elements (such as function names and class names), filenames, pathnames, directory names, HTML tags, and any text that must be typed on the screen. (*`Monospace italic font`* is used for placeholders embedded in code.)
- *Italic type* is used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
- **Boldface type** is used for glossary terms.

About the Sample Code

This book contains sample Java code for LDAP clients. This code was tested with the JDK 1.1.8 on a machine running Microsoft Windows NT version 4.0 SP4.

1

Introduction to LDAP and the Netscape Directory SDK for Java

Chapter

1

Understanding LDAP

This chapter explains the LDAP protocol and the concepts behind LDAP.

LDAP (Lightweight Directory Access Protocol) is the Internet directory protocol. Developed at the University of Michigan at Ann Arbor in conjunction with the Internet Engineering Task Force, LDAP is a protocol for accessing and managing directory services.

The chapter is organized in the following sections:

- "How Directory Services Work"
- "How LDAP Servers Organize Directories"
- "How LDAP Clients and Servers Work"
- "Understanding the LDAP v3 Protocol"
- "For More Information"

If you are already familiar with LDAP, you can skip ahead to Chapter 2, "Using the Netscape Directory SDK for Java."

How Directory Services Work

A directory consists of entries containing descriptive information. For example, a directory might contain entries describing people or network resources, such as printers or fax machines.

The descriptive information is stored in the attributes of the entry. Each attribute describes a specific type of information. For example, attributes describing a person might include the person's name (common name, or cn), telephone number, and email address.

The entry for `Barbara Jensen` might have the following attributes:

```
cn: Barbara Jensen
mail: babs@ace.com
telephoneNumber: 555-1212
roomNumber: 3995
```

An attribute can have more than one value. For example, a person might have two common names (a formal name and a nickname) or two telephone numbers:

```
cn: Jennifer Jensen
cn: Jenny Jensen
mail: jen@ace.com
telephoneNumber: 555-1213
telephoneNumber: 555-2059
roomNumber: 3996
```

Attributes can also contain binary data. For example, attributes of a person might include the JPEG photo of the person or the voice of the person recorded in an audio file format.

A directory service is a distributed database application designed to manage the entries and attributes in a directory. A directory service also makes the entries and attributes available to users and other applications. The Netscape Directory Server is an example of a directory service.

For example, a user might use the directory service to look up someone's telephone number. Another application might use the directory service to retrieve a list of email addresses.

LDAP is a protocol defining a directory service and access to that service. LDAP is based on a client-server model. LDAP servers provide the directory service, and LDAP clients use the directory service to access entries and attributes.

An example of an LDAP server is the Netscape Directory Server, which manages and provides information about users and organizational structures of users, such as groups and departments. Examples of LDAP clients might include the HTTP gateway to the Netscape Directory Server, Netscape Navigator, and Netscape Communicator. The gateway uses the directory service to find, update, and add information about users.

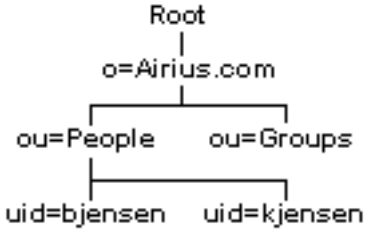
How LDAP Servers Organize Directories

Because LDAP is intended to be a global directory service, data is organized hierarchically, starting at a root and branching down into individual entries.

At the top level of the hierarchy, entries represent larger organizations. Under these larger organizations in the hierarchy might be entries for smaller organizations. The hierarchy might end with entries for individual people or resources.

Figure 1.1 illustrates an example of a hierarchy of entries in an LDAP directory service.

Figure 1.1 A hierarchy of entries in the directory



Each entry is uniquely identified by a distinguished name. A distinguished name consists of a name that uniquely identifies the entry at that hierarchical level (for example, `bjensen` and `kjensen` are different user IDs that identify different entries at the same level) and a path of names that trace the entry back to the root of the tree.

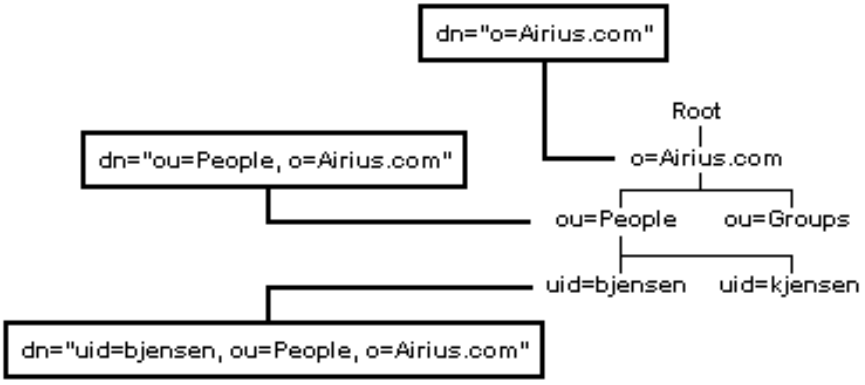
For example, this might be the distinguished name for the `bjensen` entry:

```
uid=bjensen, ou=People, o=Airius.com
```

Here, `uid` represents the user ID of the entry, `ou` represents the organizational unit in which the entry belongs, and `o` represents the larger organization in which the entry belongs.

The following diagram shows how distinguished names are used to identify entries uniquely in the directory hierarchy.

Figure 1.2 An example of a distinguished name in the directory



The data stored in a directory can be distributed among several LDAP servers. For example, one LDAP server at Airius.com might contain entries representing North American organizational units and employees, while another LDAP server might contain entries representing European organizational units and employees.

Some LDAP servers are set up to refer requests to other LDAP servers. For example, if the LDAP server at Airius.com receives a request for information about an employee in a Pacific Rim branch, that server can refer the request to the LDAP server at the Pacific Rim branch. In this way, LDAP servers can appear to be a single source of directory information. Even if an LDAP server does not contain the information you request, the server can refer you to another server that does contain the information.

How LDAP Clients and Servers Work

In the LDAP client-server model, LDAP servers (such as the Netscape Directory Server) make information about people, organizations, and resources accessible to LDAP clients. The LDAP protocol defines operations that clients use to search and update the directory. An LDAP client can perform these operations, among others:

- searching for and retrieving entries from the directory
- adding new entries to the directory
- updating entries in the directory
- deleting entries from the directory
- renaming entries in the directory

For example, to update an entry in the directory, an LDAP client submits the distinguished name of the entry with updated attribute information to the LDAP server. The LDAP server uses the distinguished name to find the entry and performs a modify operation to update the entry in the directory.

To perform any of these LDAP operations, an LDAP client needs to establish a connection with an LDAP server. The LDAP protocol specifies the use of TCP/IP port number 389, although servers may run on other ports.

The LDAP protocol also defines a simple method for authentication. LDAP servers can be set up to restrict permissions to the directory. Before an LDAP client can perform an operation on an LDAP server, the client must authenticate itself to the server by supplying a distinguished name and password. If the user identified by the distinguished name does not have permission to perform the operation, the server does not execute the operation.

Understanding the LDAP v3 Protocol

Many LDAP servers support version 2 of the LDAP protocol. This version of the protocol is specified in RFC 1777 (you can find a copy of this RFC at `http://www.ietf.org/rfc/rfc1777.txt`).

The most recent proposed standard is version 3 of the LDAP protocol, which is specified in RFC 2251 (you can find a copy of this RFC at `http://www.ietf.org/rfc/rfc2251.txt`). Some LDAP servers, such as the Netscape Directory Server 3.0 and later, support this newer version of the protocol.

The Netscape Directory SDK for Java 4.0 supports both of these versions of the protocol. Clients built with this SDK can interact with LDAP v2 servers and LDAP v3 servers.

The LDAP v3 protocol includes these new features:

- You can specify controls (both on the server and on the client) that extend the functionality of an LDAP operation.
- You can request the server to perform extended operations (beyond the standard LDAP operations).
- You can use Simple Authentication and Security Layer (SASL) mechanisms to authenticate to the directory.
- Servers have DSEs (DSA-specific entries, where a DSA is a directory server) that provide information including the versions of the LDAP protocol supported, a list of the controls, extended operations, and SASL mechanisms supported by the server, and the naming contexts of the server (specifying the portion of the directory tree managed by this server).
- Servers make their schemas available to clients. (You can get a directory server's schema from the root DSE.)
- Both client and severs can support data in UTF-8 format. Clients can now request and receive data that is tagged with language information.

For More Information

Chapter 1, "Welcome to the Directory Server," in the *Netscape Directory Deployment Guide*. provides a more detailed introduction to the LDAP protocol and directory services.

Chapter

2

Using the Netscape Directory SDK for Java

This chapter describes the Lightweight Directory Access Protocol (LDAP) Java classes and the Netscape Directory SDK for Java.

The chapter contains the following sections:

- "Understanding the LDAP Java Classes"
- "Getting Started with the Netscape Directory SDK for Java"

Understanding the LDAP Java Classes

The Netscape Directory SDK for Java includes the LDAP Java classes, which you use to build LDAP clients. The LDAP Java classes allow you to write applets and applications that can connect to LDAP servers and perform standard LDAP operations (for example, you can search for entries or add, update, or delete entries).

The classes are organized in the following packages:

- `netscape.ldap` contains the main LDAP Java classes, including classes that allow you to connect to an LDAP server, manipulate entries and attributes, and retrieve search results.

- `netscape.ldap.beans` contains the LDAP JavaBeans. You can use these Beans in a development environment such as Sun's Bean Development Kit (BDK).
- `netscape.ldap.ber.stream` contains the LDAP Java classes that implement the Basic Encoding Rules (BER) for transfer syntax. For more information on BER, see ISO/IEC 8825 at `http://www.iso.ch/`.
- `netscape.ldap.controls` contains the LDAP Java classes that implement specific LDAP v3 controls. These include controls to request server-side sorting and persistent searches.
- `netscape.ldap.util` contains utility classes, such as classes to parse LDIF data and filters that allow regular expression matching.
- `com.netscape.sasl` contains the interfaces and classes that you can use to enable your client to authenticate by using a SASL mechanism.
- `com.netscape.jndi` contains Netscape's LDAP service provider and its dependent classes. This JNDI implementation is discussed further in Chapter 15, "Using the JNDI Service Provider."

Typically, clients execute the methods in the Netscape Directory SDK for Java synchronously. All LDAP operations block until they are completed (with the exception of the search method, which can return information before all the results have been received).

An asynchronous interface is also provided for circumstances requiring low-level interaction with an LDAP server. The asynchronous interface is discussed more fully in Chapter 17, "Using the Asynchronous Interface."

Subsequent chapters in this manual explain how to use these LDAP Java classes.

Getting Started with the Netscape Directory SDK for Java

This section covers the following topics:

- Getting and Installing the SDK

- Exploring the SDK
- Preparing to Use the SDK
- Writing Applets with the SDK
- Checking the Version of Classes from an Applet
- Working with the LDAP JavaBeans
- Using the Classes in JavaScript

Getting and Installing the SDK

You can get the Netscape Directory SDK for Java from Netscape's DevEdge web site at the following location:

`http://developer.netscape.com/tech/directory/index.html`

The SDK is available as a GNU-zipped tar file for UNIX, a self-extracting executable for Windows, and a StuffIt archive for the Macintosh.

To install the SDK:

1. Download the appropriate file for your operating system
2. Expand the file (it contains a number of subdirectories).

Exploring the SDK

The Netscape Directory SDK for Java contains the following directories:

- `beans`

  This directory contains the LDAP JavaBean class files, which are part of the `netscape.ldap.beans` package.

  Note that these classes are not included with Netscape Communicator. If you are writing applications or applets that use these classes, make sure to provide these classes to your users.

  This directory also contains a `makejars.bat` file and a `makejars.sh` shell script. You can use these to create JAR files for the LDAP JavaBeans.

- `examples`

  This directory contains sample source code for LDAP applications in Java. The examples are organized in different subdirectories:

  - `java` contains examples of the standard LDAP operations, such as adding an entry and searching for entries. This directory also contains examples using LDAP controls.
  - `java/beans` contains examples of using the LDAP JavaBeans.
  - `java/ldapfilt` contains an example of using an LDAP filter configuration file with the LDAP filter classes. (Note that the LDAP filter classes are not included with Netscape Communicator.)
  - `js` contains an example of using LiveConnect to create and manipulate LDAP Java objects from JavaScript. (LiveConnect is Netscape's technology for enabling communication in a single page between a variety of elements including JavaScript, HTML, plug-ins and Java applets.)

- `packages`

  This directory contains the following JAR files:

  - `ldapjdk.jar` - This JAR file contains the classes in the `netscape.ldap`, `netscape.ldap.controls`, `netscape.ldap.util`, and `com.netscape.sasl` packages.
  - `ldapfilt.jar`- This JAR file contains the filter classes in the `netscape.ldap.util` package and the `com.oroinc.text.regex` package.

  `com.oroinc.text.regex` is the OROMatcher™ regular expression package from ORO Java Software. If you want to use the OROMatcher package separately (not through the Netscape Directory SDK for Java classes), you must obtain a license to use the OROMatcher package from ORO Java Software. (You can also obtain the OROMatcher documentation directly from ORO.)

- `tools`

  This directory contains Java classes that are similar to the command-line utilities provided with the Netscape Directory Server 4.0 and the Netscape Directory C SDK. (Note that the Java tools do not support all of the command-line arguments available with these other utilities.)

Preparing to Use the SDK

Before compiling any applets or applications, make sure to add the following to your `CLASSPATH` environment variable:

- the `packages/ldapjdk.jar` file, which contains the main LDAP Java classes
- the `packages/ldapfilt.jar` file, if you plan to use any of the LDAP Java filter classes
- the `classes` directory, if you plan to use any of the LDAP JavaBean classes

Writing Applets with the SDK

In Netscape Communicator, an applet can connect to servers on hosts other than the host that served the applet. This capability is part of the new signed applet security framework.

To take advantage of this capability, your applet class (the class making an LDAP connection) must be signed. Your applet class needs to request certain special rights before connecting to other servers.

The rest of this section summarizes the steps that you need to take to enable your applet to connect to other LDAP servers.

1. Get a certificate from your organization's certificate authority (if your organization issues certificates internally) or from a third-party certificate authority, such as RSA, Verisign, or ATT.

   Users should have the certificate from the certificate authority in the Communicator certificate database.

2. Create a JAR file with your classes and have them signed.

   To do this, you can use the JAR file management tools, which are available at:

   `http://developer.netscape.com/software/`

   Additional documentation on Netscape Object Signing technology is available at:

   `http://developer.netscape.com/docs/manuals/signedobj/`

3. Add the following line to your applet code in the thread where you invoke `LDAPConnection.connect`:

   ```
   PrivilegeManager.enablePrivilege("UniversalConnect");
   ```

   At this point in the code, the user of your applet will be prompted with a dialog box identifying the author of the signed class and asking permission to grant the right to access the LDAP server. The user can either allow access for this time only or forever.

If you want to test the ability for your applet to connect to other hosts (other than the originating one) without using object signing and certificates (for example, in case you want to test your applet while waiting for your certificate to be issued), you can configure Netscape Communicator to bypass this check.

First, exit Communicator (or make sure that it is not running). Then, add the following line to the `prefs.js` file:

```
user_pref("signed.applets.codebase_principal_support", true);
```

Checking the Version of Classes from an Applet

Netscape Communicator 4.0 and more recent versions include a copy of the LDAP Java classes. Different versions of Communicator have different versions of the classes.

The following table lists some of the different versions of the classes that have been released up to this point in time.

Table 2.1 Recently released versions of the LDAP Java classes

Version	Description
3.03	Released with the Netscape Directory SDK for Java 3.0
3.1	Released with the Netscape Directory SDK for Java 3.1.
4.0	Available at: `http://developer.netscape.com/software/ldap/`

Version 3.03 and beyond of the LDAP Java classes include new classes such as `LDAPCache` and `LDAPSchema`. If you plan to use these classes in an applet, make sure to check the version of the LDAP Java classes available in the browser.

To get the version number, invoke the `LDAPConnection.getProperty` method and pass in the constant `LDAPConnection.LDAP_PROPERTY_SDK`. The version number is a `Float` type. For example:

```
...
Float sdkVersion = ( Float )myConn.getProperty( myConn.LDAP_PROPERTY_SDK
);
System.out.println( "LDAP Java Classes version: " + sdkVersion );
...
```

For information on the differences between the LDAP Java classes in this release and in previous releases, see the release notes at:

`http://developer.netscape.com/tech/directory/index.html?content=java40relnotes.html`

Working with the LDAP JavaBeans

The Netscape Directory SDK for Java includes a set of LDAP JavaBeans that you can use in a design environment, such as Sun Microsystems' BeanBox or Symantec's Visual Cafe.

These Beans are part of the `netscape.ldap.beans` package. The class files are located in the `beans` directory.

The following JavaBeans are included with the SDK:

- The `LDAPGetEntries` Bean allows you to search the directory and get an array of the DNs found by the search. You can use the properties of this Bean to specify the search criteria. The `getEntries` method performs the search and sets the `Result` property to the array of DNs found.
- The `LDAPGetProperty` Bean allows you to find an entry in the directory and get the values of a specified attribute in that entry. You can use the properties of this Bean to specify the search criteria. The `getProperty` method performs the search and sets the `Result` property to the array of the string values of the specified attribute.
- The `LDAPIsMember` Bean determines is a user is a member of a group (the user and group can be specified as properties of this Bean). The `isMember` method sets the `Result` property to the string "Y" or "N" to indicate if the user is a member.
- The `LDAPSimpleAuth` Bean authenticates to an LDAP server. The `authenticate` method performs the authentication and sets the `Result` property to the string "Y" or "N" to indicate whether or not authentication was successful.

The `netscape.ldap.beans` package also includes the following classes:

- The `LDAPBasePropertySupport` class is a base class that the other Bean classes extend. This class specifies accessor methods that are inherited by the other Bean classes.
- The `DisplayString` class extends the `java.awt.TextArea` class and is provided to help you display the results of some of the Beans.

Before you use the Beans in a design environment, make sure to set your `CLASSPATH` environment variable to include the LDAP Java classes. For example:

- If you are using Sun Microsystems BeanBox utility, make sure that in the `run.bat` file, the makefile, or the current shell or console window, the `CLASSPATH` environment variable includes the path to the `ldapjdk.jar` file. For example:

```
set CLASSPATH=classes;D:\netscape\ldapjdk\packages\ldapjdk.jar
```

- If you are using Symantec Visual Cafe 2.x, make sure that in the `VisualCafe\bin\sc.ini` file, the `CLASSPATH` entry includes the path to the `ldapjdk.jar` file. For example:

```
CLASSPATH=.;...<other_paths>;
D:\NETSCAPE\LDAPJDK\PACKAGES\LDAPJDK.JAR
```

Next, set up your design environment to include the JAR files under the `beans` directory of the Netscape Directory SDK for Java. For example:

- If you are using the BeanBox utility, copy the JAR files to the `BDK/jars` directory. When you start up the BeanBox, the LDAP JavaBeans should be loaded in automatically.
- If you are using Visual Cafe, choose the View | Component Library menu command to display the Component Library window. After making sure that you have a project currently open, choose the Insert | Component Into Library menu command and select a JAR file containing an LDAP JavaBean. This should add the Bean to your component library.

Note that the LDAP JavaBeans are not visible Beans; they are not UI components.

The Beans have a `Debug` property that you can set to specify that you want debugging information printed out. This information is printed to standard output, so if you are testing the applet in BeanBox or in appletviewer, debug messages are printed out to the console or shell window.

Using the Classes in JavaScript

Using Netscape Communicator's LiveConnect capabilities, you can use the LDAP Java classes from within JavaScript code in an HTML page. (LiveConnect enables communication between JavaScript and Java applets in a page and between JavaScript and plug-ins loaded on a page.)

To see an example of how you can do this, see the sample JavaScript code in the HTML file in the `examples/js` directory. For information on LiveConnect, see the chapter on LiveConnect in the Netscape JavaScript Guide (available at `http://developer.netscape.com/docs/manuals/js/client/jsguide/index.htm`) and Netscape Tech Note TN-JSCR-03-9707 (available at `http://developer.netscape.com/docs/technote/javascript/liveconnect/liveconnect_rh.html`).

Chapter

3

Quick Start

This chapter provides a simple example of an LDAP client written with the Netscape Directory SDK for Java.

The chapter contains the following sections:

- “Understanding the Sample Client”
- “Sample Code”

Understanding the Sample Client

The following is the source code for a program that retrieves the full name ("cn"), last name ("sn"), email address ("mail"), and telephone number ("telephoneNumber") of Barbara Jensen. You can find this program in the `GetAttrs.java` file in the `examples/java` directory.

Basically, the example does the following:

1. Creates a new `LDAPConnection` object, which represents the connection to the LDAP server.
2. Connects to the server.

3. Performs a search that retrieves a single entry, identified by its DN. To do this, the search criteria is set up so that:

   - The base distinguished name (the starting point of the search) is the entry "uid=bjensen, ou=People, o=Airius.com"
   - The scope of the search is `LDAPConnection.SCOPE_BASE`, which only includes the base DN
   - The search filter is "`(objectclass=*)`", which finds anything that matches. Since the scope narrows the search down to a single entry, the search filter does not need to be used.

   (Note that performing a search for a single entry is equivalent to invoking the `LDAPConnection.read` method.)

   - The string array `attrNames` contains the attributes that the search will return.
   - The `attrsOnly` property is set to `false`, which means that the names and values of the specified attributes will be returned.

4. Iterates through the enumerated search results to retrieve and print the values of the cn, sn, mail, and telephoneNumber attributes. This iteration also allows the client to obtain multiple instances of a single attribute (two telephone numbers, for example).

5. Disconnects from the server.

Before you compile the sample client, replace "localhost" and "389" with the hostname and port number of the LDAP server that you are using. Also, make sure that the `packages/ldapjdk.jar` file is in your `CLASSPATH`.

Sample Code

```
import netscape.ldap.*;
import java.util.*;
public class GetAttrs {
   public static void main( String[] args )
   {
      LDAPConnection ld = null;
      LDAPEntry findEntry=null;
```

```
int status = -1;
try {
   ld = new LDAPConnection();
   /* Connect to server */
   String MY_HOST = "localhost";
   int MY_PORT = 389;
   ld.connect( MY_HOST, MY_PORT );
   String ENTRYDN = "uid=bjensen, ou=People, o=Airius.com";
   String[] attrNames = {
      "cn",       /* Get canonical name(s) (full name) */
      "sn",       /* Get surname(s) (last name) */
      "mail",     /* Get email address(es) */
      "telephonenumber" };/* Get telephone number(s) */
   LDAPSearchResults res = ld.search( ENTRYDN,
         LDAPConnection.SCOPE_BASE, "objectclass=*",
         attrNames, false );
   /* Loop on results until finished; will only be one! */
   while ( res.hasMoreElements() ) {
      /* Next directory entry, really only one at most */
      LDAPEntry findEntry = null;
      try {
         findEntry = res.next();
      /* If the next result is a search reference,
         print the LDAP URLs in the reference. */
      } catch ( LDAPReferralException e ) {
         System.out.println( "Search reference: " );
         LDAPUrl refUrls[] = e.getURLs();
         for (int i=0; i<refUrls.length; i++) {
            System.out.println( "\t" + refUrls[i].getURL() );
         }
         continue;
      } catch ( LDAPException e ) {
         System.out.println( "Error: " + e.toString() );
         continue;
      }
      /* Get the attributes of the entry */
      LDAPAttributeSet findAttrs=findEntry.getAttributeSet();
      Enumeration enumAttrs = findAttrs.getAttributes();
      /* Loop on attributes */
      while ( enumAttrs.hasMoreElements() ) {
         LDAPAttribute anAttr =
            (LDAPAttribute)enumAttrs.nextElement();
         String attrName = anAttr.getName();
         if ( attrName.equals( "cn" ) )
            System.out.println( "Full name:" );
         else if ( attrName.equals( "sn" ) )
            System.out.println( "Last name (surname):" );
         else if ( attrName.equals( "mail" ) )
            System.out.println( "Email address:" );
         else if ( attrName.equals( "telephonenumber" ) )
```

```
                    System.out.println( "Telephone number:" );
                 /* Loop on values for this attribute */
                 Enumeration enumVals = anAttr.getStringValues();
                 while ( enumVals.hasMoreElements() ) {
                    String aVal = ( String )enumVals.nextElement();
                    System.out.println( "\t" + aVal );
                 }
              }
           }
        }
        catch( LDAPException e ) {
           System.out.println( "Error: " + e.toString() );
        }
        /* Done, so disconnect */
        if ( (ld != null) && ld.isConnected() ) {
           try {
               ld.disconnect();
           } catch ( LDAPException e ) {
              System.out.println( "Error: " + e.toString() );
           }
        }
        System.exit(status);
     }
  }
```

2

Writing Clients with the Netscape Directory SDK for Java

Chapter 9 Comparing Values in Entries

This chapter explains how to compare the value of an attribute in an entry against a specified value.

Chapter 10 Working with LDAP URLs

This chapter describes what LDAP URLs are and explains how to use LDAP URLs to search and retrieve data from the directory.

Chapter

4

Writing an LDAP Client

This chapter describes the general process of writing an LDAP client. The chapter covers the procedures for connecting to an LDAP server, authenticating, requesting operations, and disconnecting from the server.

The chapter includes the following sections:

- "Overview: Designing an LDAP Client"
- "Creating a Connection and Setting Preferences"
- "Connecting to the LDAP Server"
- "Binding and Authenticating to an LDAP Server"
- "Performing LDAP Operations"
- "Closing the Connection to the Server"

The next chapter, "Using the LDAP Java Classes," also includes important information on using the LDAP Java classes to write an LDAP client.

Overview: Designing an LDAP Client

With the Netscape Directory SDK for Java, you can write an application or applet that can interact with an LDAP server. The following procedure outlines the typical process of communicating with an LDAP server. Follow these steps when writing your LDAP client:

1. Create a new `LDAPConnection` object, and set any preferences that you want applied to all LDAP operations. (See “Creating a Connection and Setting Preferences” for details.)

2. Connect to an LDAP server. (See “Connecting to the LDAP Server” for details.)

3. If necessary, bind to the LDAP server. If you intend to use any of the LDAP v3 features (such as controls or extended operations), specify the version of LDAP supported by your client. (See “Binding and Authenticating to an LDAP Server” for details.)

4. Perform the operations (for example, search the directory or modify entries in the directory). (See “Performing LDAP Operations” for details.)

5. When you are done, disconnect from the LDAP server. (See “Closing the Connection to the Server” for details.)

The following is a simple example of an LDAP client that follows the steps listed above to search a directory. The client connects to the LDAP server running on the local machine at port 389, authenticates as the DN "uid=bjensen,ou=People,o=Airius.com", searches the directory for entries with the last name "Jensen" ("sn=Jensen"), and prints out the DNs of any matching entries.

```
import netscape.ldap.*;
import java.util.*;
public class SimpleExample {
   public static void main( String[] args )
   {
      /* Step 1: Create a new connection. */
      LDAPConnection ld = new LDAPConnection();

      try {
         /* Step 2: Connect to an LDAP server. */
         ld.connect( "localhost", LDAPv2.DEFAULT_PORT );
```

```
            /* Step 3: Authenticate to the server.
               If you do not specify a version number,
               this method authenticates your client
               as an LDAP v2 client (not LDAP v3). */
            ld.authenticate( "uid=bjensen,ou=People,o=Airius.com",
               "hifalutin" );

            /* Step 4: Perform your LDAP operations. */
            /* Search for all entries with the last name "Jensen". */
            LDAPSearchResults results = ld.search( "o=Airius.com",
               LDAPv2.SCOPE_SUB, "(sn=Jensen)", null, false );
            /* Print the DNs of the matching entries. */
            while ( results.hasMoreElements() ) {
               LDAPEntry entry = null;
               try {
                  entry = results.next();
                  System.out.println( entry.getDN() );
               } catch ( LDAPReferralException e ) {
                  System.out.println( "Search references: " );
                  LDAPUrl refUrls[] = e.getURLs();
                  for ( int i=0; i < refUrls.length; i++ ) {
                     System.out.println( "\t" + refUrls[i].getURL() );
                  }
                  continue;
               } catch ( LDAPException e ) {
                  System.out.println( "Error: " + e.toString() );
                  continue;
               }
            }
         } catch( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
         }

         /* Step 5: Disconnect from the server when done. */
         try {
            ld.disconnect();
         } catch( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
            System.exit(1);
         }
         System.exit(0);
      }
   }
```

The rest of this chapter explains how to connect to the server, bind to the server, perform LDAP operations, and disconnect from the server.

Creating a Connection and Setting Preferences

The first step in writing an LDAP client is creating an `LDAPConnection` object. This object represents the connection to an LDAP server.

For example:

```
LDAPConnection ld = new LDAPConnection();
```

Note If you plan to connect to the LDAP server over a Secure Sockets Layer (SSL), you need to specify a class that implements SSL sockets. For details, see Chapter 12, "Connecting Over SSL".

In addition, the object also contains preferences for the LDAP session (for example, whether or not referrals are automatically followed).

To get or set the value of a preference, invoke the `getOption` method or the `setOption` method. In both of these methods, you can use the `option` parameter to specify the preference that you want to work with.

For a complete list of the preferences that you can get and set, see the documentation on the `getOption` method or the `setOption` method.

Connecting to the LDAP Server

To connect to an LDAP server, use the `connect` method of the `LDAPConnection` object. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
```

`DEFAULT_PORT` specifies the default LDAP port, port 389.

You can also specify a list of LDAP servers to attempt to connect to. If the first LDAP server in the list does not respond, the client will attempt to connect to the next server in the list.

Use a space-delimited list of the host names as the first argument of the `connect` method. If the server is not using the default LDAP port (port 389), specify the port number in *`hostname`*:*`portnumber`* format. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap1.airius.com ldap2.airus.com:3890
   ldap3.airius.com:39000", LDAPv2.DEFAULT_PORT );
```

Binding and Authenticating to an LDAP Server

When connecting to the LDAP server, your client may need to send a bind operation request to the server. This is also called binding to the server.

An LDAP bind request contains the following information:

- the LDAP version of the client
- the DN that the client is attempting to authenticate as
- the method of authentication that should be used
- the credentials to be used for authentication

Your client should send a bind request to the server in the following situations:

- You want to authenticate to the server.

  For example, you may want to add or modify entries in the directory, which requires you to authenticate as a user with access privileges.

- You are connecting to an LDAP v2 server.

  LDAP v2 servers typically require clients to bind before any operations can be performed. (Note that the Netscape Directory Server 1.0x is an LDAP v2 server but does not require clients to bind.)

LDAP clients can also bind as an anonymous clients to the server (for example, the LDAP server may not require authentication if your client is just searching the directory).

This chapter explains how to set up your client to bind to an LDAP server. Topics covered here include:

- “Understanding Authentication Methods”
- “Using Simple Authentication”

- "Binding Anonymously"
- "Specifying the LDAP Version"
- "Authenticating with the connect Method"

Understanding Authentication Methods

When binding to an LDAP server, you can authenticate your client using one of three methods: Simple authentication, Certificate-based client authentication (over SSL), and the Simple Authentication and Security Layer (SASL). This section will discuss the differences between these methods.

Simple Authentication

With simple authentication, your clients provides the distinguished name of the user and the user's password to the LDAP server.

For more information on simple authentication, see "Using Simple Authentication".

You can also use this method to bind as an anonymous client by providing `null` values as the user's distinguished name and password (as described in "Binding Anonymously").

Certificate-Based Client Authentication (over SSL)

With certificate-based client authentication, your client sends its certificate to the LDAP server. The certificate identifies your LDAP client.

For more information on using certificate-based client authentication, see Chapter 12, "Connecting Over SSL".

Simple Authentication and Security Layer (SASL)

SASL is described in RFC 2222. Some LDAP v3 servers (including the Netscape Directory Server 3.0 and later) support authentication through SASL.

For more information on using SASL mechanisms for authentication, see Chapter 14, "Using SASL Authentication".

Using Simple Authentication

If you plan to use simple authentication, use the `authenticate` method of the `LDAPConnection` object. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
ld.authenticate( "uid=bjensen,ou=People,o=Airius.com", "hifalutin" );
```

If you are binding to the Netscape Directory Server 3.0 or later, the server may send back special controls to indicate that your password has expired or will expire in the near future. For more information on these controls, see “Using Password Policy Controls”.

Binding Anonymously

In some cases, you may not need to authenticate to the LDAP server. For example, if you are writing a client to search the directory (and if users don’t need special access permissions to search), you might not need to authenticate before performing the search operation.

However, in the LDAP v2 protocol, the server still expects the client to send a bind request, even if the operation does not require the client to authenticate itself. (In the LDAP v3 protocol, the server no longer expects the client to send a bind request in this type of situation.)

In this kind of situation, use the `authenticate` method and specify `null` for the DN and password. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
ld.authenticate( null, null );
```

Specifying the LDAP Version

As part of the bind request sent to the server, the client includes the version of the LDAP protocol that it supports. By default, clients built with the Netscape Directory SDK for Java identify themselves as LDAP v2 clients.

If you want to use any of the LDAP v3 features (such as controls or extended operations), you need to identify your client as an LDAP v3 client.

(Before you set up your client to use LDAP v3 or request any LDAP v3 features, you should first verify that the server supports LDAP v3.)

To identify your client as an LDAP v3 client, do one of the following:

- Specify version 3 when invoking the `authenticate` method. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
ld.authenticate( 3, null, null );
```

- Invoke the `setOption` method of the `LDAPConnection` object to set the `LDAPv2.PROTOCOL_VERSION` preference to 3, then invoke the `authenticate` method. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
ld.setOption( LDAPv2.PROTOCOL_VERSION, 3 );
ld.authenticate( null, null );
```

Authenticating with the connect Method

The `connect` method of the `LDAPConnection` object has a signature that allows you to authenticate and specify the LDAP version supported by your client.

You can specify all of this information using one method, rather than invoking several methods. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( 3, "ldap.airius.com", DEFAULT_PORT,
   "uid=bjensen,ou=People,o=Airius.com", "hifalutin" );
```

Performing LDAP Operations

Once you initialize a session with an LDAP server and complete the authentication process, you can perform LDAP operations (such as searching the directory, adding new entries, updating existing entries, and removing entries), provided that the server's access control allows you to request these operations.

To perform LDAP operations, invoke these methods of the `LDAPConnection` object:

- To search for entries in the directory, use the `search` method (see "Searching the Directory" for details).
- To retrieve a single entry in the directory, use the `read` method (see "Searching the Directory" for details).
- To determine whether an attribute contains a certain value, use the `compare` method (see "Comparing the Value of an Attribute" for details).
- To add entries to the directory, use the `add` method (see "Adding a New Entry" for details).
- To modify entries in the directory, use the `modify` method (see "Modifying an Entry" for details).
- To delete entries from the directory, use the `delete` method (see "Deleting an Entry" for details).
- To rename entries in the directory, use the `rename` method (see "Changing the Name of an Entry" for details).

Closing the Connection to the Server

When you have finished performing all necessary LDAP operations, you need to close the connection to the LDAP server.

Use the `disconnect` method of the `LDAPConnection` object to disconnect from the LDAP server. For example:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
...
/* Authenticate and perform LDAP operations on the server. */
...
ld.disconnect();
```

Chapter

5

Using the LDAP Java Classes

This chapter covers some of the general LDAP Java classes that are commonly used when writing LDAP clients.

This chapter covers the following topics:

- “Getting Information About the SDK”
- “Handling Exceptions”
- “Handling Referrals”
- “Using an In-Memory Cache”
- “Cloning a Connection”
- “Manipulating Distinguished Names”

Getting Information About the SDK

You can get version information about the Netscape Directory SDK for Java that you are using (for example, the version of the LDAP Java classes or the highest version of the LDAP protocol that it supports).

To get this information, use the `getProperty` method of the `LDAPConnection` object. For a complete list of the SDK properties you can get, see the documentation on the `getProperty` method.

For example, the following section of code lists the properties of the SDK:

```
LDAPConnection ld = new LDAPConnection();
try {
   System.out.println( "LDAP Java Classes Version: " +
      (Float)ld.getProperty(LDAPConnection.LDAP_PROPERTY_SDK) );
   System.out.println( "Highest version of LDAP supported: " +
      (Float)ld.getProperty(LDAPConnection.LDAP_PROPERTY_PROTOCOL) );
   System.out.println( "Authentication methods supported: " +
      (String)ld.getProperty(LDAPConnection.LDAP_PROPERTY_SECURITY) );
} catch (LDAPException e) {
   System.out.println( "Could not get SDK properties." );
   System.out.println( "Error: " + e.toString() );
}
...
```

For information on the different versions of the LDAP Java classes, see the documentation on `LDAP_PROPERTY_SDK`.

Note that although a `setProperty` method is listed as one of the available methods, there are currently no properties that you can set. Invoking this method throws an `LDAPException`.

Handling Exceptions

In the LDAP protocol, the success or failure of an operation is specified by an LDAP result code sent back to the client. (For example, the result code 0 indicates that the operation was successful, and a non-zero result code usually indicates that an error occurred.)

The following sections explain more about exceptions in the LDAP Java classes.

- “Getting Information About the Error”
- “Getting the Error Message”

Getting Information About the Error

In the LDAP Java classes, when an error occurs, an `LDAPException` is thrown. (Note that referrals cause an `LDAPReferralException` to be thrown. For details, see “Handling Referrals”.)

An `LDAPException` contains the following information:

- the LDAP result code for the error that occurred
- a message containing any additional information about the error from the server

If the error occurred because an entry specified by a DN could not be found, the `LDAPException` also contains the DN of the "closest matching entry" that could be found.

To get information from the `LDAPException`, use one of the following methods:

- To get the string representation of the exception, use the `toString` method.
- To get the LDAP result code, use the `getLDAPResultCode` method.
- To get any additional information sent by the server, use the `getLDAPErrorMessage` method.
- To get the closest matching DN (in cases where your client specified a DN to a non-existent entry), use the `getMatchedDN` method.

Note that you can also get the error message describing the LDAP result code by using the `errorCodeToString` method. (See the section “Getting the Error Message” for details.)

The Directory SDK for Java’s JavaDocs include a listing and descriptions of the different LDAP result codes. For information on using these JavaDocs, see “Where to Find Reference Information”.

The following section of code gets and prints information about an `LDAPException`:

```
...
try {
/* Attempt to perform an LDAP operation here. */
```

```
} catch( LDAPException e ) {
   /* Get and print the result code and any other info. */
   int resultCode = e.getLDAPResultCode();
   String serverInfo = e.getLDAPErrorMessage();
   System.out.println( "LDAP Result Code: " + resultCode );
   if ( serverInfo != null ) {
      System.out.println( "Additional Info: " + serverInfo );
   }
   /* If the exception was thrown because an entry
       was not found, print the DN of the closest entry found. */
   switch( resultCode ) {
      case LDAPException.NO_SUCH_OBJECT:
      case LDAPException.ALIAS_PROBLEM:
      case LDAPException.INVALID_DN_SYNTAX:
      case LDAPException.ALIAS_DEREFERENCING_PROBLEM:
         String matchedDN = e.getMatchedDN();
         if ( matchedDN != null ) {
            System.out.println( "Closest Entry: " + matchedDN );
         }
         break;
      default:
         break;
   }
}
...
```

Getting the Error Message

To get the error message for an LDAP result code, use the `errorCodeToString` method. For example:

```
...
try {
/* Attempt to perform an LDAP operation here. */
} catch( LDAPException e ) {
   /* Get and print the error message. */
   int resultCode = e.getLDAPResultCode();
   System.out.println( "Error: " + e.errorCodeToString( resultCode ) );
}
...
```

Error messages corresponding to each LDAP result code are stored in files in the following location:

`dir_root/netscape/ldap/error/ErrorCodes_locale_name.props`

where *dir_root* is a directory that is in your CLASSPATH, and *locale_name* is the language and country (concatenated and delimited by an underscore) of the current locale.

For example:

```
netscape/ldap/error/ErrorCodes_en_US.props
netscape/ldap/error/ErrorCodes_en.props
```

A default error messages file is provided in the following location:

```
netscape/ldap/error/ErrorCodes.props
```

The result codes and their corresponding descriptions have the following format in the file:

resultcode=errormessage

For example:

```
0=Success
1=Operations error
2=Protocol error
3=Timelimit exceeded
4=Sizelimit exceeded
5=Compare false
6=Compare true
...
```

When you invoke the errorCodeToString method of an LDAPException, the method retrieves the error messages file for the specified locale (or the default locale, if no locale is specified) and reads the error message from the file.

The method searches for the error messages file in the following locations in this order (if no file is found, the method tries to find the next file in this list):

```
netscape/ldap/error/ErrorCodes_language_country.props
netscape/ldap/error/ErrorCodes_language.props
netscape/ldap/error/ErrorCodes.props
```

Handling Referrals

If an LDAP server receives a request for a DN that is not under its directory tree, it can refer clients to another LDAP server that may contain that DN. This is known as a referral.

This section explains how to set up your LDAP client to handle referrals automatically. The following topics are covered:

- “Understanding Referrals”
- “Enabling or Disabling Referral Handling”
- “Limiting Referral Hops”
- “Binding When Following Referrals”

Understanding Referrals

Suppose an LDAP server has a directory that starts under "o=Airius.com". If your client sends the server a request to modify the entry with the DN "uid=bjensen,ou=People,o=AiriusWest.com" (an entry that is not under "o=Airius.com"), one of the following may occur:

- If the server is not configured to send a referral, an `LDAPException` is thrown with the LDAP result code `LDAPException.NO_SUCH_OBJECT`.
- If the server is configured to refer you to another LDAP server, the server sends a referral back to your client. Depending on how your LDAP client is configured one of the following may occur:
  - If your client handles referrals automatically, your client connects to the LDAP server specified in the referral and requests to modify the entry. (The client binds anonymously to that server. To bind as a specific user, see the section “Binding When Following Referrals”.)
  - If your client does not handle referrals automatically, an `LDAPReferralException` is thrown. You can get the LDAP URL specified in the referral by catching the exception and invoking the `getURLs` method.

  By default, clients built with the Netscape Directory SDK for Java are configured to follow referrals automatically.

Another concept similar to a referral is a search reference. A search reference is an entry with the object class "referral". The "ref" attribute of this object contains an LDAP URL that identifies another LDAP server.

When your client searches a subtree of the directory that contains search references, the server returns a mix of matching entries and search references. As you iterate through the enumeration of search results, if you encounter a search reference and your client does not handle referrals automatically, an `LDAPReferralException` is thrown.

Enabling or Disabling Referral Handling

By default, clients built with the Netscape Directory SDK for Java automatically follow these referrals to other servers.

To change the way your client handles referrals, use the `setOption` method of the `LDAPConnection` object (to change the behavior for all LDAP operations) or the `setReferrals` method of the `LDAPConstraints` object (to change the behavior for a specific search request):

- To prevent the client from automatically following referrals, pass `LDAPv2.REFERRALS` and `false` as arguments to the `setOption` method, or pass `false` as the argument to the `setReferrals` method.
- If you want the client to automatically follow referrals again, pass `LDAPv2.REFERRALS` and `true` as arguments to the `setOption` method, or pass `true` as the argument to the `setReferrals` method.

Limiting Referral Hops

As a preference for the connection (or as a search constraint for specific search operations), you can specify the maximum number of referral hops that should be followed in a sequence of referrals. This is called the referral hop limit.

For example, suppose you set a limit of 2 referral hops. If your client is referred from LDAP server A to LDAP server B, from LDAP server B to LDAP server C, and from LDAP server C to LDAP server D, your client is being referred 3 times in a row, and it will not follow the referral to LDAP server D because this exceeds the referral hop limit.

If the referral hop limit is exceeded, an `LDAPReferralException` is thrown.

To set the referral hop limit, use the `LDAPv2.REFERRALS_HOP_LIMIT` preference with the `setOption` method of the `LDAPConnection` object (to change the behavior for all LDAP operations) or the `setHopLimit` method of the `LDAPConstraints` object (to change the behavior for a specific search request).

By default, the maximum number of referral hops is 10.

Binding When Following Referrals

If the connection is set up so that referrals are always followed (see "Enabling or Disabling Referral Handling" for more information), the LDAP server that you connect to may refer you to another LDAP server. By default, the client binds anonymously (no username or password are specified) when following referrals.

If you want your client to authenticate to the LDAP server that it is referred to, you need to specify a way to get the DN and password to be used for authentication. You need to define a class that implements the `LDAPRebind` interface. Then, you specify an object of this new class using the `LDAPv2.REFERRALS_REBIND_PROC` preference with the `setOption` method of the `LDAPConnection` object (to set this for all LDAP operations) or the `setRebindProc` method of the `LDAPConstraints` object (to change the behavior for a specific search request).

The `LDAPRebind` interface specifies a `getRebindAuthentication` method that returns an `LDAPRebindAuth` object. The `getRebindAuthentication` method and the `LDAPRebindAuth` object it returns are used to get the DN and password for authentication.

The following steps explain how this works:

1. The LDAP server sends a referral back to the client. The referral contains an LDAP URL that identifies another LDAP server.
2. The client creates a new `LDAPConnection` object.
3. The client connects to the host and port specified in the LDAP URL.

4. Using the `getRebindProc` method to find your object (the one that implements the `LDAPRebind` interface), the client invokes the `getRebindAuthentication` method, passing in the host and port specified in the LDAP URL.

5. The `getRebindAuthentication` method in your object returns an `LDAPRebindAuth` object.

6. The client invokes the `getDN` and `getPassword` methods of the returned `LDAPRebindAuth` object to get the DN and password to use for authentication.

7. The client invokes the `authenticate` method of the `LDAPConnection` object and passes the DN and password to authenticate to the server.

Basically, you need to define the following:

- a class that implements the `LDAPRebind` interface
- a `getRebindAuthentication` that takes host name and port number and creates an `LDAPRebindAuth` object that specifies the DN and password to use for authentication

Using an In-Memory Cache

The Netscape Directory SDK for Java includes an `LDAPCache` class that allows you to create an in-memory cache of search results for your client. When send a search request and receive results, the search request and its results are cached. The next time your client issues the same search request, the results are read from the cache.

This section covers the following topics:

- “How the Cache Operates”
- “Setting Up an In-Memory Cache”
- “Caching Requests by Base DN”
- “Sharing a Cache Between Connections”
- “Flushing the Cache”

- “Getting Cache Statistics”

How the Cache Operates

Each item in the cache represents a search request and its results. When you create the cache, you can specify the maximum size of the cache and the maximum amount of time that an item can be cached:

- When an item's age exceeds that time limit, the item is removed from the cache. The cache is checked once a minute for expired items.
- If adding a new item will cause the cache to exceed its maximum size, items are removed from the cache to make space for the new item. Items are removed on a FIFO (first in, first out) basis.

Each item is uniquely identified by the search criteria, which includes:

- the host name and port number of the LDAP server
- the base DN of the search
- the search filter
- the scope of the search
- the attributes to be returned in the search results
- the DN used to authenticate the client when binding to the server
- the LDAP v3 controls specified in the search request

Once a search request is cached, if your client performs the same search again, the results are read from the cache instead of from the server.

Note that if any part of a search request differs from a cached search request (for example, if a different DN is used when authenticating to the server or if the search request specifies that a different set of attributes should be returned), the results are not read from the cache and the search request is sent to the server.

Finally, when creating the cache, you can specify a list of the base DNs in search requests that you want to cache. For example, if you specify "ou=People, o=Airius.com" as a base DN to cache, your client caches search requests where the base DN is "ou=People, o=Airius.com".

Setting Up an In-Memory Cache

To set up a cache for your connection, do the following:

1. Construct a new `LDAPCache` object which represents the cache.
2. Invoke the `setCache` method of an `LDAPConnection` object to associate the cache with the connection.

For example, the following section of code creates a cache with the maximum size of 1MB. Items in the cache expire and are automatically removed after 1 hour.

```
...
import netscape.ldap.*;
import java.util.*;
...
LDAPConnection ld = null;
int status = -1;
String MY_HOST = "localhost";

try {
   ld = new LDAPConnection();

   /* Create a cache for the connection */
   int MAX_TIME_CACHED = 3600;
   int MAX_SIZE = 1000000;
   LDAPCache myCache = new LDAPCache( MAX_TIME_CACHED, MAX_SIZE );
   ld.setCache( myCache );

   /* Connect to server */
   ld.connect( MY_HOST, LDAP_PORT );
   ...
} catch( LDAPException e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

Caching Requests by Base DN

If you do not want all search requests cached, you can specify a set of base DNs for search requests that you want cached in the `LDAPCache` constructor.

For example, the following section of code constructs a cache that only tracks search requests that specify the base DNs "ou=People, o=Airius.com" and "ou=Groups,o=Airius.com".

```
...
/* Create a cache for the connection */
int MAX_TIME_CACHED = 3600;
int MAX_SIZE = 1000000;
String [] BASE_DN_CACHED = {"ou=People,o=Airius.com",
   "ou=Groups,o=Airius.com"};
LDAPCache myCache = new LDAPCache( MAX_TIME_CACHED, MAX_SIZE,
   BASE_DN_CACHED );
...
```

Sharing a Cache Between Connections

You can also share the same in-memory cache among different connections. To do this:

1. Invoke the `getCache` method of an `LDAPConnection` object to get the `LDAPCache` object associated with it.
2. Invoke the `setCache` method of a different `LDAPConnection` object to associate the retrieved `LDAPCache` object with it.

Note that when you clone an `LDAPConnection` object, the new object automatically shares the same `LDAPCache` object with the original object.

Flushing the Cache

To flush items from the cached, invoke the `flushEntries` method of the `LDAPCache` object. You can either flush selected items from the cache or all items from the cache.

To flush selected items from the cache, specify the base DN of the search requests that you want to flush. To do this, you can specify a DN and search scope as arguments to the `flushEntries` method. If the base DN of a cached search request falls within the scope you've specified, the search request is flushed from the cache.

For example, the following section of code flushes selected search requests from the cache. If the base DN of a search request falls under the "ou=People,o=Airius.com" subtree, the item is removed from the cache.

```
...
import netscape.ldap.*;
import java.util.*;
...
LDAPCache myCache;
...
/* Perform search requests. */
...
/* Flush items from the cache. Flush the
   search requests that have base DNs under the
   "ou=People, o=Airius.com" subtree. */
myCache.flushEntries( "ou=People, o=Airius.com",
   LDAPConnection.SCOPE_SUB );
...
```

To flush all items from the cache, pass `null` for the first argument of the `flushEntries` method:

```
myCache.flushEntries( null, 0 );
```

Getting Cache Statistics

You can invoke the following methods of the `LDAPCache` object to get statistics on the cache:

- To get the total amount of available space (in bytes) left in the cache, invoke the `getAvailableSize` method.
- To get the array of base DNs of the search requests to be cached, invoke the `getBaseDNs` method.
- To get the total number of items that have been flushed from the cache (not including items flushed when invoking the `flushEntries` method), invoke the `getNumFlushes` method.

- To get the maximum size of the cache (in bytes), invoke the `getSize` method.
- To get the maximum amount of time that an item can be cached (in seconds), get the `getTimeToLive` method.

You can also get a running count of the number of requests that access the cache:

- To get the total number of requests for retrieving items from the cache, invoke the `getTotalOperation` method.
- To get the total number of requests that retrieved an item from the cache, invoke the `getNumHits` method.
- To get the total number of requests that failed to find and retrieve an item from the cache, invoke the `getNumMisses` method.

For example, the following section of code gets and prints cache statistics.

```
...
import netscape.ldap.*;
import java.util.*;
...
import netscape.ldap.*;
import java.util.*;
...
LDAPConnection ld;
...
if ( (ld != null) && ld.isConnected() ) {
   LDAPCache connCache = ld.getCache();
   if ( connCache != null ) {
      System.out.println( "Cache size:\t\t" +
         connCache.getSize()/1000 + " kbytes" );
      System.out.println( "Available:\t\t" +
         connCache.getAvailableSize()/1000 + " kbytes" );
      System.out.println( "Maximum age:\t\t" +
         connCache.getTimeToLive()/1000 + " seconds" );
      System.out.println( "Total hits:\t\t" +
         connCache.getNumHits() + " hits" );
      System.out.println( "Total misses:\t\t" +
         connCache.getNumMisses() + " misses" );
      System.out.println( "Total requests:\t\t" +
         connCache.getTotalOperations() + " requests" );
} else {
      System.out.println( "No cache associated with the connection." );
...
```

Cloning a Connection

You can create several `LDAPConnection` objects that share a single physical network connection to the LDAP server by invoking the `LDAPConnection.clone` method. (Note, however, that if you clone an object before a connection is made to the server, the cloned object does not share the same connection as the original object.)

Each clone can disconnect from the server without affecting the connection for the other clones. The network connection remains open until all clones have disconnected or gone out of scope.

If the clone disconnects or reconnects, it is completely dissociated from the source object and other clones.

Note the following:

- A cloned `LDAPConnection` object has a separate set of session preferences and constraints (in other words, it is associated with its own `LDAPSearchConstraints` object). Changing an option or search constraint will only affect requests issued using that object.

  For example, suppose an `LDAPConnection` object allows a maximum of 100 results to be returned from a search. If you clone an `LDAPConnection` object and change the maximum number of results in the clone from 100 to 200, the initial object will still allow only 100 results to be returned.

- All clones are authenticated to the server as the same user. If you reauthenticate to the server as a different user, the current clone will be disconnected and will connect separately to the server. (The `LDAPConnection` object will no longer be a clone of another `LDAPConnection` object.)

Manipulating Distinguished Names

A distinguished name (DN) uniquely identifies an entry in the directory tree. You can get the DN for an entry (see “Getting Distinguished Names from the Results”) or specify a DN to read an entry from the directory (see “Reading an Entry”).

The Netscape Directory SDK for Java contains a utility class, `LDAPDN`, that provides methods for manipulating DNs.

Getting the Components of a Distinguished Name

If you want to access individual components of a distinguished name or a relative distinguished name, invoke the `LDAPDN.explodeDN` method or the `LDAPDN.explodeRDN` method.

Both functions return an array of strings representing the individual components of the distinguished name.

You can specify whether or not you want the names of the components included in the array by using the `notypes` parameter.

- Set `notypes` to `false` if you want to include component names in the array.

  ```
  LDAPDN.explodeDN( "uid=bjensen, ou=People, o=Airius.com", false )
  ```

  The method returns this array:

  ```
  { "uid=bjensen", "ou=People", "o=Airius.com" }
  ```

- Set `notypes` to `true` if you don't want to include the component names in the array.

  ```
  LDAPDN.explodeDN( "uid=bjensen, ou=People, o=Airius.com", true )
  ```

  The method returns this array:

  ```
  { "bjensen", "People", "Airius.com" }
  ```

Chapter

6

Searching the Directory

This chapter explains how to use the LDAP Java classes to search the directory and retrieve entries. The chapter also describes how to get attributes and attribute values from an entry.

This chapter covers the following topics:

- “Overview: Searching with the LDAP Java Classes”
- “Sending a Search Request”
- “Getting the Search Results”
- “Sorting the Search Results”
- “Abandoning a Search”
- “Example: Searching the Directory”
- “Reading an Entry”
- “Listing Subentries”

Overview: Searching with the LDAP Java Classes

In the Netscape Directory SDK for Java, searches are represented by objects of the following classes:

- You can send a search request by invoking the `search` method of the `LDAPConnection` object.
- You can specify a set of search constraints (such as the maximum number of results to return or the maximum amount to time allowed for a search) by using an `LDAPSearchConstraints` object.
- You can either specify different parts of the search criteria in separate arguments, or you can construct an `LDAPUrl` object to specify the search criteria.
- The server returns the search results to the LDAP Java classes, which represents the results as an `LDAPSearchResults` object.
- You can search for a single entry by invoking the `read` method of the `LDAPConnection` object.

The rest of this chapter explains how to use these classes and methods to search a directory.

Sending a Search Request

To search the directory, use the `search` method of the `LDAPConnection` object. The search results are returned in the form of an `LDAPSearchResults` object.

```
public LDAPSearchResults search(String base, int scope,
   String filter, String attrs[], boolean attrsOnly,
   LDAPSearchConstraints cons) throws LDAPException
```

You need to specify the following information as arguments to the `search` method:

- `base` specifies the starting point in the directory, or the base DN (an entry where to start searching)

To search entries under “o=Airius.com”, the base DN is “o=Airius.com”. See “Specifying the Base DN and Scope” for details.

- `scope` specifies the scope of the search (which entries you want to search)

  You can narrow the scope of the search to search only the base DN, entries at one level under the base DN, or entries at all levels under the base DN. See “Specifying the Base DN and Scope” for details.

- `filter` specifies a search filter (what to search for)

  A search filter can be as simple as “find entries where the last name is Jensen” or as complex as “find entries that belong to Dept. #17 and whose first names start with the letter F.” See “Specifying a Search Filter” for details.

- `attrs` and `attrsOnly` specify the type of information that you want return (which attributes you want to retrieve)

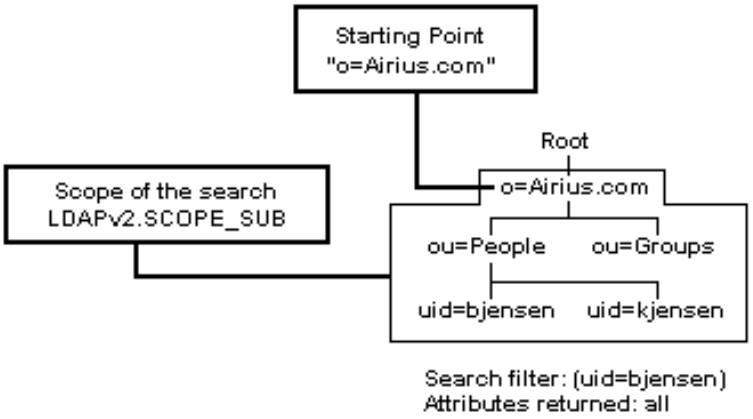
  For example, you can return only email addresses and phone numbers, or you can set up a search to return all attributes in an entry. You can also specify that you only want to return the names of attributes, not the values. See “Specifying the Attributes to Retrieve” for details.

- `cons` specifies the search constraints that you want applied to this search

  For example, you can set up a set of search constraints that differs from the default set of constraints. See “Setting Search Preferences” for details.

Figure 6.1 illustrates how search criteria works.

Figure 6.1 Search criteria of an LDAP search operation



You can also specify the criteria in the form of an LDAP URL. An LDAP URL allows you to specify the hostname and port number of the LDAP server that you want to search. (If you want to search a different LDAP server than the one you are connected to, you can invoke the `search` method and specify an LDAP URL in the form of an `LDAPUrl` object.)

For details on using LDAP URLs for searches, see Chapter 10, “Working with LDAP URLs”.

Specifying the Base DN and Scope

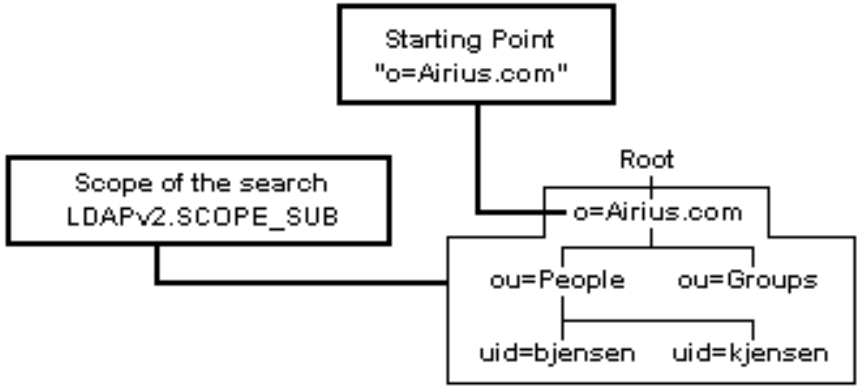
When sending a search request, you need to specify the base DN and scope of the search to identify the entries that you want searched.

The base DN (the `base` argument) is the DN of the entry that serves as the starting point of the search.

To specify the scope of the search, you pass one of the following values as the `scope` parameter:

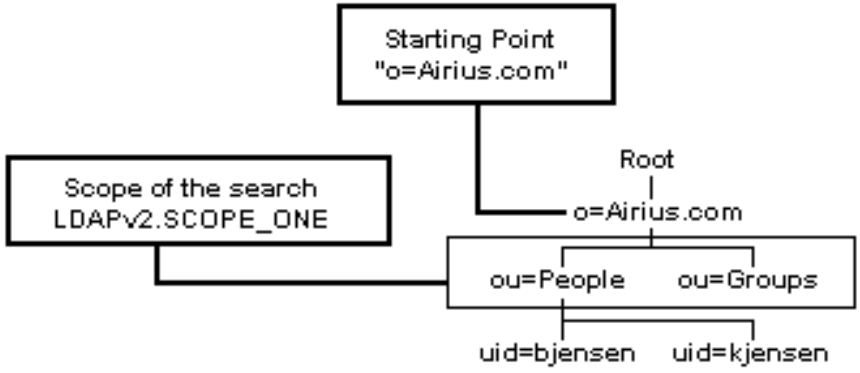
- `LDAPv2.SCOPE_SUB` searches the `base` entry and all entries at all levels below the `base` entry (as illustrated in Figure 6.2).

Figure 6.2 Example of a search with the scope LDAPv2.SCOPE_SUB



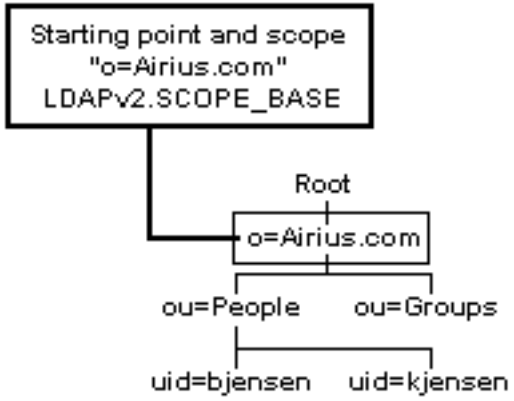
- LDAPv2.SCOPE_ONE searches all entries at one level below the base entry (as illustrated in Figure 6.3). The base entry is not included in the search. Use this setting if you just want a list of the entries under a given entry. (See “Listing Subentries” on page 89 for an example.)

Figure 6.3 Example of a search with the scope LDAPv2.SCOPE_ONE



- LDAPv2.SCOPE_BASE searches only the base entry. Use this setting if you just want to read the attributes of the base entry (as illustrated in Figure 6.4). (See “Reading an Entry” on page 87 for an example.)

Figure 6.4 Example of a search with the scope LDAPv2.SCOPE_BASE



Specifying a Search Filter

When you search the directory, you use a search filter to define the search. Here is the basic syntax for a search filter:

(*attribute operator value*)

Here is a simple example of a search filter:

```
(cn=Barbara Jensen)
```

In this example, `cn` is the attribute, `=` is the operator, and `Barbara Jensen` is the value. The filter finds entries with the common name `Barbara Jensen`

For a listing of valid attributes that you can use in your search filter, see the documentation for the LDAP server. (For information on the attributes in the schema for the Netscape Directory Server, see the Administrator's Guide.)

Table 6.1 lists the valid operators you can use.

Table 6.1 Basic operators for search filters

Operator	Description	Example
=	Returns entries whose attribute is equal to the value.	(cn=Barbara Jensen) finds the entry “cn=Barbara Jensen”
>=	Returns entries whose attribute is greater than or equal to the value.	(sn >= jensen) finds all entries from “sn=jensen” to “sn=z...”
<=	Returns entries whose attribute is less than or equal to the value.	(sn <= jensen) finds all entries from “sn=a...” to “sn=jensen”
=*	Returns entries that have a value set for that attribute.	(sn =*) finds all entries that have the sn attribute.
~=	Returns entries whose attribute value approximately matches the specified value. Typically, this is an algorithm that matches words that sound alike.	(sn ~= jensen) finds the entry “sn = jansen”

Note that when comparing values containing letters, the letter a is less than the value z. For example, the following filter finds all entries with last names beginning with a through jensen:

```
(sn<=jensen)
```

Using Boolean operators and parentheses, you can combine different sets of conditions. Here is the syntax for combining search filters:

```
( boolean_operator (filter1)(filter2)(filter3) )
```

Table 6.2 lists the valid boolean operators you can use.

Table 6.2 Boolean operators for search filters

Boolean Operator	Description
&	Returns entries matching all specified filter criteria.

Table 6.2 Boolean operators for search filters

Boolean Operator	Description
|	Returns entries matching one or more of the filter criteria.

!	Returns entries for which the filter is not true.You can only apply this operator to a single filter. For example, you can use:
	`(!(filter))`
	but not:
	`(!(filter1)(filter2))`

For example, you can use this filter to search for all entries with the last name `Jensen` or the last name `Johnson`:

```
(|(sn=Jensen)(sn=Johnson))
```

You can also include wildcards to search for entries that start with, contain, or end with a given value. For example, you can use this filter to search for all entries whose names begin with the letter F:

```
(givenName=F*)
```

Specifying the Attributes to Retrieve

Using the `attrs` argument, you can either retrieve all attributes in entries returned by the search, or you can specify the attributes that you want returned in the search results. For example, you can specify that you want to return the following attributes:

- To return selected attributes, pass an array of the attribute names as the `attrs` argument. For example, to return only email addresses and phone numbers, pass the array `{"mail", "telephoneNumber"}` as the `attrs` argument.
- To return all attributes in an entry, pass `null` as the `attrs` argument.
- To return none of the attributes from an entry, pass `LDAPv3.NO_ATTRS` as the `attrs` argument.

Note that if you plan to sort the results on your client (see "Sorting the Search Results"), you need to return the attributes that you plan to use for sorting. For example, if you plan to sort by email address, make sure that the mail attribute is returned in the search results.

Some attributes are used by servers for administering the directory. For example, the `creatorsName` attribute specifies the DN of the user who added the entry.These attributes are called operational attributes.

Servers do not normally return operational attributes in search results unless you specify the attributes by name. For example, if you pass `null` for the `attrs` argument to retrieve all of the attributes in entries found by the search, the operational attribute `creatorsName` will not be returned to your client. You need to explicitly specify the `creatorsName` attribute in the `attrs` argument.

To return all attributes in an entry and selected operational attributes, pass a string array containing `LDAPv3.ALL_USER_ATTRS` and the names of the operational attributes as the `attrs` argument.

The following table lists some of the operational attributes and the information they contain.

Table 6.3 Information available in operational attributes

Attribute Name	Description of Values
`createTimestamp`	The time the entry was added to the directory.
`modifyTimestamp`	The time the entry was last modified.
`creatorsName`	Distinguished name (DN) of the user who added the entry to the directory.
`modifiersName`	Distinguished name (DN) of the user who last modified the entry.
`subschemaSubentry`	Distinguished name (DN) of the subschema entry, which controls the schema for this entry. (See "Getting Schema Information" for details.)

Setting Search Preferences

For a given search, you can apply a set of preferences that determine how the search is performed. For example, you can specify the maximum number of results to be returned or the maximum amount of time to wait for a search.

The `LDAPSearchConstraints` class represents a set of search constraints. The methods of this class allow you to get and set the constraints.

The rest of this section explains how to set these constraints. The following topics are discussed:

- Setting Preferences for All Searches
- Overriding Preferences on Individual Searches
- Configuring the Search to Wait for All Results
- Setting Size and Time Limits

Setting Preferences for All Searches

The `LDAPConnection` object (which represents a connection to the LDAP server) is associated with a default set of search constraints. These constraints apply to all searches performed over the connection.

- To get the default set of search constraints for the connection, you can use the `getSearchConstraints` method.
- To get or set any of the search constraints individually, you can use the `getOption` method and the `setOption` method.

For example, if you want to specify the maximum number of results returned, you can either set this constraint for the connection:

```
LDAPConnection ld = new LDAPConnection();
ld.connect( "ldap.airius.com", LDAPv2.DEFAULT_PORT );
ld.setOption( LDAPv2.SIZELIMIT, new Integer( 100 ) );
```

Overriding Preferences on Individual Searches

To override the default set of search constraints for a given search request, construct your own `LDAPSearchConstraints` object and pass it to the `search` method of the `LDAPConnection` object.

You can modify a copy of the existing search constraints and pass the modified set of constraints to the `search` method. Invoke the `getSearchConstraints` method of the `LDAPConnection` object to get the default set of constraints for that connection, then invoke the `clone` method of the `LDAPSearchConstraints` object to make a copy of the set.

Configuring the Search to Wait for All Results

By default, the `search` method of the `LDAPConnection` object does not block until all results are received. Instead, the `search` method returns as soon as one of the results has been received.

If you want to change this so that the `search` method blocks until all results are returned, you can do one of the following:

- Use the `setOption` method of the `LDAPConnection` object to set the `LDAPv2.BATCHSIZE` preference to 0.
- Pass a 0 to the `setBatchSize` method of the `LDAPSearchConstraints` object to change this for a particular set of search constraints.

Note that in either case, you still need to invoke the `next` method of the returned `LDAPSearchResults` object to retrieve each individual result.

Setting Size and Time Limits

By default, when you search the directory from a client built with the Netscape Directory SDK for Java, the maximum number of entries to return is set to 1000, and there is no maximum time limit for waiting on an operation to complete.

To change these default values, you can do one of the following:

- Use the `setOption` method of the `LDAPConnection` object to set the `LDAPv2.SIZELIMIT` and `LDAPv2.TIMELIMIT` preferences.
- Use the `setMaxResults` method and the `setTimeLimit` method of the `LDAPSearchConstraints` object to change this for a particular set of search constraints.

Setting the size limit or time limit may cause an `LDAPException` to be thrown if the size limit or time limit is exceeded:

- If the size limit is exceeded, the server returns an `LDAPException.SIZE_LIMIT_EXCEEDED` result code.
- If the time limit is exceeded, the server returns an `LDAPException.TIME_LIMIT_EXCEEDED` result code.)

Note that the smallest unit supported by the Netscape Directory Server is seconds, not milliseconds. Since the Netscape Directory SDK for Java allows you to specify the time limit in milliseconds, the value that you specify will be rounded to the nearest second by the Directory Server.

Example of Sending a Search Request

The following section of code searches for all entries with the last name (or surname) "Jensen" in the Airius.com organization. The search retrieves the names and values of the `cn`, `mail`, and `telephoneNumber` attributes.

```
...
LDAPConnection ld = null;
try {
   /* Create a new LDAPConnection object. */
   ld = new LDAPConnection();

   /* Connect and bind to the server. */
   String HOSTNAME = "localhost";
   ld.connect( HOSTNAME, DEFAULT_PORT, null, null );

   /* Specify the search criteria. */
   String baseDN = "o=Airius.com";
   int searchScope = LDAPv2.SCOPE_SUB;
   String searchFilter = "(sn=Jensen)";
   String getAttrs[] = {"cn", "mail", "telephoneNumber"};

   /* Send the search request. */
   LDAPSearchResults res = ld.search( baseDN, searchScope,
      searchFilter, getAttrs, false );
   ...
} catch( LDAPException e ) {
   System.out.println( "Error: " + e.toString );
}
...
```

Getting the Search Results

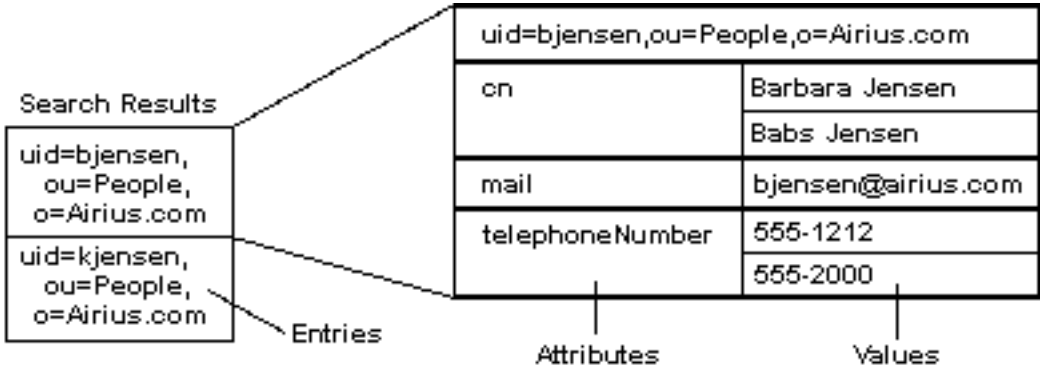
When you invoke the `search` method of an `LDAPConnection` object to search the directory, the method returns the search results in the form of an `LDAPSearchResults` object.

The search results consist of an enumeration of entries, which are represented by `LDAPEntry` objects. The search results can also include smart referrals (also known as a search reference) and exceptions.

Each entry contains a set of attributes, which are represented by `LDAPAttributeSet` objects. Individual attributes are represented by `LDAPAttribute` objects. Each attribute has a set of values that you can get.

Figure 6.5 illustrates the relationship between entries, attributes, values, and search results.

Figure 6.5 Search results in terms of entries, attributes, and values



The rest of this section explains the steps for getting data from the search results:

- For information on getting individual entries from the results, see “Getting Entries from the Results”.
- For information on getting the DN of an entry, see “Getting Distinguished Names from the Results”.
- For information on getting the attributes of an entry, see “Getting Attributes from an Entry”.

- For information on getting the values of an attribute, see “Getting the Name and Values of an Attribute”.

Getting Entries from the Results

The `LDAPSearchResults` object represents the results of the search. These results can include entries found by the search, search references, and result codes. (If your LDAP client receives an `ADMIN_LIMIT_EXCEEDED`, `TIME_LIMIT_EXCEEDED`, or `SIZE_LIMIT_EXCEEDED` result code from the server, the Netscape Directory SDK for Java adds an exception for this result code to the search results.)

To get entries from the `LDAPSearchResults` object, you can either invoke the `next` method or the `nextElement` method.

- When you invoke the `next` method, if the next item in the search results is an entry, the method returns an `LDAPEntry` object.

  If the next item is a search reference, one of the following can occur:

  - If referrals are not followed automatically, or if the referral hop limit has been exceeded, an `LDAPReferralException` is thrown.

  - If referrals are followed automatically and if the referral hop limit has not been exceeded, the LDAP Java classes follow the referral and retrieve the entry for you. (The method creates a new connection to the server specified in the referral and attempts to retrieve the entry from that server.)

  For information about referrals and search references, see “Handling Referrals”.

  If the next item is an LDAP result code such as `ADMIN_LIMIT_EXCEEDED`, `TIME_LIMIT_EXCEEDED`, or `SIZE_LIMIT_EXCEEDED`, the LDAP Java classes throw an `LDAPException`.

- When you invoke the `nextElement` method, the method returns an object that you must cast. The object is an `LDAPEntry` object, an `LDAPReferralException`, or an `LDAPException`.

As you iterate through the search results, you can invoke the `hasMoreElements` method to determine if you have reached the end of the search results.

For example:

```
...
LDAPConnection ld = null;
try {
   /* Create a new LDAPConnection object. */
   ld = new LDAPConnection();
   ...
   /* Send the search request. */
   LDAPSearchResults res = ld.search( baseDN, searchScope,
      searchFilter, getAttrs, false );

   /* Iterate through the results until finished. */
   while ( res.hasMoreElements() ) {

      /* Get the next entry in the results. */
      LDAPEntry findEntry = null;
      try {
         findEntry = res.next();

      /* If it is a search reference,
         print the LDAP URLs in the reference. */
      } catch ( LDAPReferralException e ) {
         System.out.println( "Search references: " );
         LDAPUrl refUrls[] = e.getURLs();
         for ( int i=0; i < refUrls.length; i++ ) {
            System.out.println( "\t" + refUrls[i].getUrl() );
         }
         continue;
      } catch ( LDAPException e ) {
         System.out.println( "Error: " + e.toString() );
         continue;
      }
      ...
   }
   ...
} catch ( LDAPException e ) {
   ...}
```

Getting Distinguished Names from the Results

To get the distinguished name of an `LDAPEntry` object, invoke the `getDN` method. This method returns a string. For example:

```
...
LDAPEntry nextEntry = res.next();
```

```
String nextDN = nextEntry.getDN();
...
```

Note that although the `netscape.ldap` package includes an `LDAPDN` class, you typically do not construct objects of this class to represent DNs. The `LDAPDN` class is mainly a utility class that provides methods for manipulating string DNs. (See "Getting the Components of a Distinguished Name" for details.)

Getting Attributes from an Entry

To get the set of attributes in an `LDAPEntry` object, invoke the `getAttributeSet` method. This method returns an `LDAPAttributeSet` object.

For example:

```
...
LDAPEntry nextEntry = res.next();
LDAPAttributeSet entryAttrs = nextEntry.getAttributeSet();
...
```

To get individual attributes from an `LDAPAttributeSet` object, invoke the `getAttributes` method. This method returns an enumeration of attributes. You can then iterate through the elements in this enumeration to retrieve individual `LDAPAttribute` objects.

For example:

```
...
/* Get the set of attributes for an entry. */
LDAPAttributeSet entryAttrs = nextEntry.getAttributeSet();

/* Get an enumeration of those attribute. */
Enumeration enumAttrs = entryAttrs.getAttributes();

/* Loop through the enumeration to get each attribute. */
while ( enumAttrs.hasMoreElements() ) {
   LDAPAttribute nextAttr =
      (LDAPAttribute)enumAttrs.nextElement();
   ...
}
...
```

To determine the number of attributes in the `LDAPAttributeSet` object, invoke the `size` method.

You can also retrieve a specific attribute from the entry or from the attribute set:

- To get a specific attribute from an `LDAPEntry` object, invoke the `getAttribute` method.
- To get a specific attribute from an `LDAPAttributeSet` object by invoking the `getAttribute` method.

Both methods return an `LDAPAttribute` object.

For example:

```
...
LDAPEntry nextEntry = res.next();
LDAPAttribute anAttr = nextEntry.getAttribute( "cn" );
...
```

Getting the Name and Values of an Attribute

To get the name of an `LDAPAttribute` object, invoke the `getName` method. For example:

```
...
LDAPAttribute nextAttr =
   (LDAPAttribute)enumAttrs.nextElement();
String attrName = nextAttr.getName();
...
```

To get the values in an `LDAPAttribute` object, you can use the following methods:

- To get the string values, invoke the `getStringValues` method.
- To get the binary values as byte arrays, invoke the `getByteValues` method.

Both methods return an enumeration that you can iterate through to retrieve individual results. If an error occurs (for example, if you invoke `getStringValues` and the values are binary data), the methods return `null`.

You can also count the number of values in an attribute by invoking the `size` method of the `LDAPAttribute` object.

For example:

```
...
LDAPAttribute nextAttr =
```

```
   (LDAPAttribute)enumAttrs.nextElement();

/* Get and print the attribute name. */
String attrName = nextAttr.getName();
System.out.println( "\t" + attrName + ":" );

/* Iterate through the attribute's values. */
Enumeration enumVals = nextAttr.getStringValues();
if (enumVals != null) {
   while ( enumVals.hasMoreElements() ) {
      String nextValue = ( String )enumVals.nextElement();
      System.out.println( "\t\t" + nextValue );
   }
}
...
```

Sorting the Search Results

With the Netscape Directory SDK for Java, you can sort the search results in two ways:

- You can specify that the LDAP server should sort the results before returning the results to your client.

  To do this, you need to send a server-side sorting control to the server. For details, see Chapter 13, "Working with LDAP Controls". Note that in the Netscape Directory Server, server-side sorting works best if you specify a filter that uses an indexed attribute.

- After you receive the results from the server, you can sort them on your client.

  To do this, you need to specify the names of the attributes that you want to use for sorting. You also need to specify whether or not the sorting is done in ascending or descending order.

You can sort the results on the client by invoking the `sort` method of the `LDAPSearchResults` object.

When invoking this method, you need to pass a comparator object, which is an object of a class that implements the `LDAPEntryComparator` interface. The Netscape Directory SDK for Java includes an `LDAPCompareAttrNames` class that implements this interface. This class specifies how entries are compared with each other and sorted.

To construct an `LDAPCompareAttrNames` object, you need to specify the attributes that you want to use for sorting and, optionally, the sort order.

For example, the following section of code sorts first by surname (“sn”) and then by common name (“cn”) in ascending order:

```
...
LDAPConnection ld = new LDAPConnection();
ld.connect( ... );
LDAPSearchResults res = ld.search( ... );
String[]  sortAttrs = {"sn", "cn"};
boolean[] ascending = {true, true};
res.sort( new LDAPCompareAttrNames(sortAttrs, ascending) );
...
```

Note Since the sorting is done by the client, the attributes used for sorting must be returned in the search results. If you are returning only a subset of attributes in the search results, make sure to include the attributes that you specify in the `LDAPCompareAttrNames` constructor.

If all search results have not yet been returned, the `sort` method will block until all results have been received.

Abandoning a Search

At any point during a search operation, you can send a request to the server to abandon (or cancel) the search. To abandon the search, use the `abandon` method of the `LDAPConnection` object. Pass in the `LDAPSearchResults` object that was returned to you when you first invoked the `search` method.

Example: Searching the Directory

The following section of code prints out the values of all attributes in the entries returned by a search.

```
...
import netscape.ldap.*;
import java.util.*;
public class Search {
   public static void main( String[] args ) {
      LDAPConnection ld = null;
      int status = -1;
```

```
try {
   ld = new LDAPConnection();

   /* Connect to the server. */
   String MY_HOST = "localhost";
   int MY_PORT = 389;
   ld.connect( MY_HOST, MY_PORT );

   /* Search for all entries with surname of Jensen. */
   String MY_FILTER = "sn=Jensen";
   String MY_SEARCHBASE = "o=Airius.com";

   /* Send the search request. */
   LDAPSearchResults res = ld.search( MY_SEARCHBASE,
      LDAPConnection.SCOPE_SUB,
      MY_FILTER, null, false );

   /* Sort the results first by last name, then by first. */
   String[]  sortAttrs = {"sn", "cn"};
   boolean[] ascending = {true, true};
   res.sort( new LDAPCompareAttrNames(sortAttrs, ascending) );

   /* Iterate through and print out the results. */
   while ( res.hasMoreElements() ) {
      /* Get the next directory entry. */
      LDAPEntry findEntry = null;
      try {
         findEntry = res.next();

      /* If the next result is a search reference,
         print the LDAP URLs in that reference. */
      } catch ( LDAPReferralException e ) {
         System.out.println( "Search references: " );
         LDAPUrl refUrls[] = e.getURLs();
         for ( int i=0; i < refUrls.length; i++ ) {
            System.out.println( "\t" + refUrls[i].getURL() );
         }
         continue;
      } catch ( LDAPException e ) {
         System.out.println( "Error: " + e.toString() );
         continue;
      }

      /* Print the DN of the entry. */
      System.out.println( findEntry.getDN() );

      /* Get the attributes of the entry */
      LDAPAttributeSet findAttrs =
         findEntry.getAttributeSet();
      Enumeration enumAttrs = findAttrs.getAttributes();
```

```
            System.out.println( "\tAttributes: " );
            /* Loop on attributes */
            while ( enumAttrs.hasMoreElements()) {
               LDAPAttribute anAttr =
                  (LDAPAttribute)enumAttrs.nextElement();
               String attrName = anAttr.getName();
               System.out.println( "\t\t" + attrName );
               /* Loop on values for this attribute */
               Enumeration enumVals = anAttr.getStringValues();
               if (enumVals != null) {
                  while ( enumVals.hasMoreElements() ) {
                     String aVal = ( String )enumVals.nextElement();
                     System.out.println( "\t\t\t" + aVal );
                  }
               }
            }
         }
         status = 0;
      } catch( LDAPException e ) {
         System.out.println( "Error: " + e.toString() );
      }
      /* Done, so disconnect */
      if ( (ld != null) && ld.isConnected() ) {
         try {
            ld.disconnect();
         } catch ( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
         }
      }
      System.exit(status);
   }
}
```

Reading an Entry

To get a single entry from the directory, use the `read` method of the `LDAPConnection` object. You can specify the DN of the entry and (optionally) the attributes that you want to retrieve (if you don't want to get all attributes of the entry back). Or you can specify an LDAP URL that identifies the entry that you want to retrieve.

You can use the same classes and methods described in "Getting Attributes from an Entry" and "Getting the Name and Values of an Attribute" to retrieve data from the entry.

The following section of code gets an entry and prints the values of its attributes.

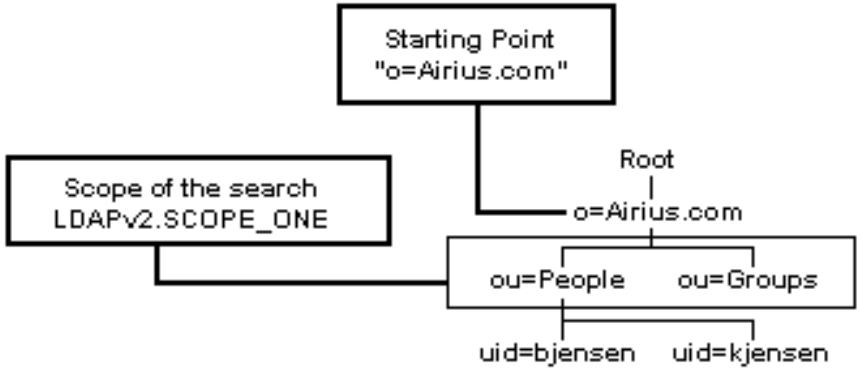
```
import netscape.ldap.*;
import java.util.*;
public class RdEntry {
   public static void main( String[] args )
   {
      LDAPConnection ld = null;
      int status = -1;
      try {
         ld = new LDAPConnection();

         /* Connect to the server. */
         String MY_HOST = "localhost";
         int MY_PORT = 389;
         ld.connect( MY_HOST, MY_PORT );

         /* Get the specified entry and all of its attributes. */
         String ENTRYDN = "uid=bjensen, ou=People, o=Airius.com";
         LDAPEntry findEntry = ld.read( ENTRYDN );

         /* Get and print the DN of the entry. */
         System.out.println( findEntry.getDN() );

         /* Get the set of attributes in the entry. */
         LDAPAttributeSet findAttrs = findEntry.getAttributeSet();
         Enumeration enumAttrs = findAttrs.getAttributes();
         System.out.println( "\tAttributes: " );

         /* Get each individual attribute. */
         while ( enumAttrs.hasMoreElements() ) {
            LDAPAttribute anAttr =
               (LDAPAttribute)enumAttrs.nextElement();

            /* Get and print the attribute name. */
            String attrName = anAttr.getName();
            System.out.println( "\t\t" + attrName );

            /* Get and print each value of this attribute. */
            Enumeration enumVals = anAttr.getStringValues();
            if (enumVals != null) {
               while ( enumVals.hasMoreElements() ) {
                  String aVal = ( String )enumVals.nextElement();
                  System.out.println( "\t\t\t" + aVal );
               }
            }
         }
      }
      catch( LDAPException e ) {
```

```
            System.out.println( "Error: " + e.toString() );
         }
         /* Done, so disconnect from the server. */
         if ( (ld != null) && ld.isConnected() ) {
            try {
               ld.disconnect();
            } catch ( LDAPException e ) {
               System.out.println( "Error: " + e.toString() );
            }
         }
         System.exit(status);
      }
   }
```

Listing Subentries

To list the subentries beneath a particular entry, set the starting point of the search to the entry, and set the scope of the search to LDAPv2.SCOPE_ONE.

Figure 6.6 Using the LDAPv2.SCOPE_ONE scope to list subentries



The following section of code performs a one-level search.

```
...
import netscape.ldap.*;
import java.util.*;
...
LDAPConnection ld = null;
try {
   ld = new LDAPConnection();
```

```
/* Connect to the server. */
String MY_HOST = "localhost";
int MY_PORT = LDAPv2.DEFAULT_PORT;
ld.connect( MY_HOST, MY_PORT );

/* Search for all entries at this level. */
String MY_FILTER = "objectclass=*";
String MY_SEARCHBASE = "o=Airius.com";
LDAPSearchResults res = ld.search( MY_SEARCHBASE,
LDAPConnection.SCOPE_ONE, MY_FILTER, null, false );

/* Loop on results until finished */
while ( res.hasMoreElements() ) {
   /* Get the next directory entry */
   LDAPEntry findEntry = null;
   try {
      findEntry = res.next();

   /* If the next result is a search reference,
      print the LDAP URLs in the reference. */
   } catch ( LDAPReferralException e ) {
      System.out.println( "Search references: " );
      LDAPUrl refUrls[] = e.getURLs();
      for ( int i=0; i < refUrls.length; i++ ) {
         System.out.println( "\t" + refUrls[i].getUrl() );
      }
      continue;
   } catch ( LDAPException e ) {
      System.out.println( "Error: " + e.toString() );
      continue;
   }

   /* Print the DN of the entry. */
   System.out.println( findEntry.getDN() );

   /* Get the attributes of the entry */
   LDAPAttributeSet findAttrs = findEntry.getAttributeSet();
   Enumeration enumAttrs = findAttrs.getAttributes();
   System.out.println( "\tAttributes: " );
   /* Loop on attributes */
   while ( enumAttrs.hasMoreElements() ) {
      LDAPAttribute anAttr = (LDAPAttribute)enumAttrs.nextElement();
      String attrName = anAttr.getName();
      System.out.println( "\t\t" + attrName );

      /* Loop on values for this attribute */
      Enumeration enumVals = anAttr.getStringValues();
      if (enumVals != null) {
         while ( enumVals.hasMoreElements() ) {
            String aVal = ( String )enumVals.nextElement();
```

```
                System.out.println( "\t\t\t" + aVal );
             }
          }
       }
    }
} catch( LDAPException e ) {
    System.out.println( "Error: " + e.toString() );
}

/* Done, so disconnect */
if ( (ld != null) && ld.isConnected() ) {
    try {
       ld.disconnect();
    } catch ( LDAPException e ) {
    System.out.println( "Error: " + e.toString() );
}
...
```

Chapter

7

Using Filter Configuration Files

This chapter explains how to use API function to work with filter configuration files. Filter configuration files can help simplify the process of selecting the appropriate search filter for a search request.

The chapter contains the following sections:

- “Understanding Filter Configuration Files”
- “Understanding the Configuration File Syntax”
- “Understanding Filter Parameters”
- “Loading a Filter Configuration File”
- “Retrieving Filters”
- “Adding Filter Prefixes and Suffixes”

Note that the LDAP filter classes use the OROMatcher™ regular expression package (from ORO Java Software), which is provided with the SDK. If you want to use the OROMatcher package separately (not through the Netscape Directory SDK for Java classes), you must obtain a license to use the OROMatcher package from ORO Java Software. (You can also obtain the OROMatcher documentation directly from ORO.)

For details, see the ORO home page at http://www.oroinc.com/.

Understanding Filter Configuration Files

Suppose that you are writing a client that allows users to search the directory. You might want to use different search filters tailored for specific types of search criteria. For example, suppose the user wants to search for this:

```
bjensen@airius.com
```

You might want to use this search filter:

```
(mail=bjensen@airius.com)
```

Similarly, suppose the search term entered by the user contains numbers, like this:

```
555-1212
```

In this case, you might want to use this search filter:

```
(telephoneNumber=555-1212)
```

Rather than write code to explicitly create the appropriate filter (based on the user's search criteria), you can include the filters in a filter configuration file.

A filter configuration file contains a list of filters that you can load and use in your searches.

Understanding the Configuration File Syntax

A filter configuration file has the following format:

```
tag
     pattern1     delimiters     filter1-1     desc1-1     [scope1]
                                 filter1-2     desc1-2     [scope2]

     pattern2     delimiters     filter2-1     desc2-1     [scope3]
     ...
```

This format is explained below:

- `tag` identifies a group of filters. You can use different tags to distinguish filters for different types of objects. For example, you can use a tag to represent filters for person entries, a tag to represent filters for organization entries, and so on:

```
"people"
   ... (filters for searching "person" entries) ...
"organization"
   ... (filters for "organization" entries) ...
```

  You can specify a tag (or part of a tag) as a parameter. The tag narrows the list of filters that the function can retrieve.

- `pattern1` and `pattern2` are regular expressions used to determine which filter is selected, based on the search criteria. For example, if you specify "^[0-9]" as the pattern for a filter, the filter is selected for all search criteria beginning with a number.

```
"people"
   "^[0-9]" ...
```

  For more information on regular expressions, consult your UNIX documentation (for example, documentation on the `ed` command contains some information on regular expressions).

- `delimiters` specifies the delimiters used to distinguish one field from another within the search criteria. For example, if the search criteria consists of a city name and state abbreviation separated by a comma, specify "," as the delimiter.

- `filter1-1`, `filter1-2`, and `filter2-1` are filters. Use `%v` to represent the search criteria. For example, to search e-mail addresses, use the filter `(mail=%v)`. During runtime, if the search criteria `bjensen@airius.com` is entered, the filter becomes `(mail=bjensen@airius.com)`.

  If the search criteria consists of a number of delimited fields (for example, a "last name, first name" format like "Jensen, Barbara"), use `%v1, %v2, ... , %vn` to represent the different fields within the search criteria. For example:

```
"people"
   "^[A-Z]*,"","  (&(sn=%v1)(givenName=%v2))
```

In this example, the delimiter is a comma. The word before the delimiter replaces `%v1` in the filter, and the word after the delimiter replaces `%v2` in the filter. If the user searches for:

```
Jensen, Barbara
```

the resulting filter is:

```
(&(sn=Jensen)(givenName=Barbara))
```

You can also specify ranges of fields. For example, to specify the values in the first three fields, use `%v1-3`. To specify values from the third field to the last field, use `%v3-`. To specify the value in the last field, use `%v$`.

- `desc1-1`, `desc1-2`, and `desc2-1` are phrases briefly describing the filters.
- `scope1`, `scope2`, and `scope3` specify the scope of each search. This field is optional and can have the values "base", "onelevel", or "subtree".

For example, the following section of a filter configuration file specifies a filter for telephone numbers and two filters for email addresses. The telephone number filter is used if the search criteria contains one or more numbers. The email filters are used if the search criteria contains an at sign (@).

```
"people"
     "^[0-9][0-9-]*$"     " "     "(telephoneNumber=*%v))"     "phone number ends with"
     "@"                  " "     "(mail=%v)"                  "email address is"
                                  "(mail=%v*)"                 "email address starts with"
```

You should specify the filters in the order that you want them to be used. For example, if you want to apply the `(mail=%v)` filter before the `(mail=%v*)` filter, make sure that the filters appear in that order.

Understanding Filter Parameters

Within a filter, you can use the following parameters:

- `%v`

  This parameter means that the entire search criteria is inserted in place of `%v`. For example, if the filter is `(mail=%v)`, entering the word `jensen` results in the filter `(mail=jensen)`.

- `%v$`

  This parameter means that the last word in the search criteria is inserted in place of `%v`. For example, if the filter is `(sn=%v$)`, entering the words `Barbara Jensen` results in the filter `(sn=Jensen)`.

- `%v`*`N`* (Note: N is a single digit between 1 and 9)

  This parameter means that the *N*th word in the search criteria is inserted in place of `%v`*`N`*. For example, if the filter is `(sn=%v2)`, entering the words `Barbara Jensen` results in the filter `(sn=Jensen)`.

- `%v`*`M-N`* (Note: M and N are single digits between 1 and 9)

  This parameter means that the sequence of the *M*th through the *N*th words in the search criteria is inserted in place of `%v`*`M-N`*. For example, if the filter is `(cn=%v1-2)`, entering the words `Barbara Jensen` results in the filter `(cn=Barbara Jensen)`.

- `%v`*`N`*`-` (Note: N is a single digit between 1 and 9)

  This parameter means that the sequence of the *N*th through the last words in the search criteria is inserted in place of `%v`*`N`*`-`. For example, if the filter is `(cn=%v2-)`, entering the words `Ms. Barbara Jensen` results in the filter `(cn=Barbara Jensen)`.

Loading a Filter Configuration File

To use a filter configuration file, you need to create an `LDAPFilterDescriptor` object. The `LDAPFilterDescriptor` constructor allows you to read in the filter configuration file from:

- a file in the local file system (you can specify the path to the file)
- a file served by a web server (you can specify a URL to the file)
- a location in memory (you can specify a `StringBuffer` containing the filter configuration information)

The following section of code reads in a filter configuration file named `ldapfilter.conf` in the current directory.

```
...
import netscape.ldap.util.*;
...
LDAPFilterDescriptor filtdesc = null;
try {
   /* Read in the filter configuration file. */
   filtdesc = new LDAPFilterDescriptor( "ldapfilter.conf" );
} catch ( Exception e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

Retrieving Filters

After loading a filter configuration file into memory, you can retrieve filters based on the search criteria. For example, if the search criteria is an e-mail address (`bjensen@airius.com`), you can have your client automatically search for this value in the `mail` attribute.

To use the filter configuration file, do the following:

1. Invoke the `LDAPFilterDescriptor` constructor to read in the filter configuration file.
2. Invoke the `getFilters` method of the `LDAPFilterDescriptor` object.

   Specify the tag of the section that you want to use in the filter configuration file.

   This method returns an `LDAPFilterList` object, which is an enumeration of `LDAPFilter` objects containing the filters for the specified search term.
3. Invoke the `next` method of the `LDAPFilterList` object to iterate through the `LDAPFilter` objects.
4. For each `LDAPFilter` object, get the filter by invoking the `getFilter` method, passing no arguments.

Note that you do not need to invoke the `setupFilter` of the `LDAPFilter` object to generate the filter. The `getFilters` method of the `LDAPFilterDescriptor` object does this already. You just need to invoke the `getFilter` method of the `LDAPFilter` object to get the generated filter.

To get the total number of filter configuration lines that match the specified search term, invoke the `numFilters` method of the `LDAPFilterList` object. Note that this number decrements each time you invoke the `next` or `nextElement` method.

The following section of code uses a filter configuration file containing the following filters:

```
"people"
     "^[0-9][0-9-]*$"     " "     "(telephoneNumber=*%v))"     "phone number ends with"
     "@"                  " "     "(mail=%v)"                  "email address is"
                                  "(mail=%v*)"                 "email address starts with"
```

The following section of code retrieves, generates, and prints filters that match the search criteria.

```
...
import netscape.ldap.util.*;
import java.util.*;

...
String searchTerm = "bjensen@airius.com";
LDAPFilterDescriptor filtdesc = null;
try {
   /* Read in the filter configuration file. */
   filtdesc = new LDAPFilterDescriptor( "ldapfilt.conf" );

   /* Get filters from the section "people" for the search term. */
   LDAPFilterList filtlist = null;
   try {
      filtlist = filtdesc.getFilters( "people", searchTerm );
   } catch ( Exception e ) {
      System.out.println( "No matching tag section or filter found." );
      System.exit(0);
   }

   int totalFilters = filtlist.numFilters();
   System.out.println( "Found " + totalFilters +
      " applicable filters.\n" );

   /* Iterate through the lines in the list. */
   while ( filtlist.hasMoreElements() ) {
      LDAPFilter filtline = filtlist.next();
      System.out.println( "Filter #" + ( totalFilters -
         filtlist.numFilters() ) );

      /* Get and print information about the selected line
         of filter configuration information. */
```

```
      System.out.println( "  Description: " +
         filtline.getDescription() );
      System.out.println( "  Line #: " + filtline.getLineNumber() );
      System.out.println( "  Matches pattern: " +
         filtline.getMatchPattern() );
      System.out.println( "  Filter template: " +
         filtline.getFilterTemplate() );
      System.out.println( "  Delimiter: " + filtline.getDelimeter() );
      System.out.println( "  Scope: " + filtline.getScope() );

      /* Get the generated filter. */
      String filterString = filtline.getFilter();
      System.out.println( "  Generated filter string: " + filterString +
         "\n" );
   }
} catch ( Exception e ) {
   System.out.println( "Error: " + e.toString() );
}
System.exit( 0 );
...
```

Suppose that the search criteria is `bjensen@airius.com`. In this case, the code prints the following output:

```
java GetFilter bjensen@airius.com
Found 2 applicable filters.

Filter #1
  Description: email address is
  Line #: 3
  Matches pattern: @
  Filter template: (mail=%v)
  Delimiter:
  Scope: subtree
  Generated filter string: (mail=bjensen@airius.com)

Filter #2
  Description: email address starts with
  Line #: 4
  Matches pattern: @
  Filter template: (mail=%v*)
  Delimiter:
  Scope: subtree
  Generated filter string: (mail=bjensen@airius.com*)
```

Adding Filter Prefixes and Suffixes

If you need to apply a filter to all searches, you can add a filter prefix and suffix to all filters (rather than add the criteria to all filters). For example, if your client searches only for person entries, you can add the following filter prefix to restrict the search:

```
(&(l=Sunnyvale)
```

Note that this filter now requires the following suffix to balance the number of parentheses:

```
)
```

For example, given the following filter:

```
(sn=Jensen)
```

you can use the filter prefix "(&(l=Sunnyvale" and the filter suffix ")" to narrow down the search to only the entries with l=Sunnyvale:

```
(&(l=Sunnyvale)(sn=Jensen))
```

You can set up the filter prefix and suffix in several ways:

- To set these for all filters generated from the filter configuration file, invoke the `setFilterAffixes` method of the `LDAPFilterDescriptor` object. See “Adding Affixes for All Filters” for an example.
- To set the prefix or suffix for a specific filter, do one of the following:
  - Invoke the `setFilterAffixes` method of the `LDAPFilter` object, then invoke the `getFilter` method, passing in the search term again. This builds the filter again, using the specified search term with the prefix and suffix. See “Adding Affixes By Using setFilterAffixes” for an example.
  - Invoke the `getFilter` method, passing in the search term, the prefix, and the suffix. See “Adding Affixes By Using getFilter” for an example.
  - Invoke the `setupFilter` method, passing the search term, the prefix, and the suffix. See “Adding Affixes By Using setupFilter” for an example.

  Note that setting the prefix and suffix for an individual filter will override any prefix or suffix set for the entire filter configuration file.

The rest of this section contains examples that illustrate these different approaches to setting prefixes and suffixes.

- “Adding Affixes for All Filters”
- “Adding Affixes By Using setFilterAffixes”
- “Adding Affixes By Using getFilter”
- “Adding Affixes By Using setupFilter”

Adding Affixes for All Filters

The following section of code loads the filter configuration file named `myfilters.conf` into memory and adds the prefix "(&(l=Sunnyvale)" and the suffix `")"` to each filter that will be retrieved from that file:

```
...
import netscape.ldap.util.*;
...
LDAPFilterDescriptor filtdesc = null;
try {
   /* Read in the filter configuration file. */
   filtdesc = new LDAPFilterDescriptor( "myfilter.conf" );

   /* Add the specified prefix and suffix to all filters. */
   String prefix = "(&(l=Sunnyvale)";
   String suffix = ")";
   filtdesc.setFilterAffixes( prefix, suffix );

   /* Get filters from the section "people" for the search term. */
   LDAPFilterList filtlist = null;
   try {
      filtlist = filtdesc.getFilters( "people", searchTerm );
   } catch ( Exception e ) {
      System.out.println( "No matching tag section or filter found." );
      System.exit(0);
   }

   /* Iterate through the lines in the list. */
   while ( filtlist.hasMoreElements() ) {
      LDAPFilter filtline = filtlist.next();

      /* Get and print each filter. */
      String filterString = filtline.getFilter();
      System.out.println( "  Generated filter string: " + filterString +
         "\n" );
```

```
   }
} catch ( Exception e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

For example, if the following search term is passed to the `LDAPFilterDescriptor.getFilters` method:

```
bjensen@airius.com
```

and the corresponding filter (without a prefix or suffix) is:

```
(mail=bjensen@airius.com)
```

the entire generated filter string (retrieved by the `LDAPFilter.getFilter` method) will be:

```
(&(l=Sunnyvale)(mail=bjensen@airius.com))
```

Adding Affixes By Using setFilterAffixes

The following section of code loads the filter configuration file named `myfilters.conf` into memory and uses the `LDAPFilter.setFilterAffixes` method to add a prefix and suffix to a generated filter:

```
...
import netscape.ldap.util.*;
...
LDAPFilterDescriptor filtdesc = null;
try {
   /* Read in the filter configuration file. */
   filtdesc = new LDAPFilterDescriptor( "myfilter.conf" );

   /* Get filters from the section "people" for the search term. */
   LDAPFilterList filtlist = null;
   try {
      filtlist = filtdesc.getFilters( "people", searchTerm );
   } catch ( Exception e ) {
      System.out.println( "No matching tag section or filter found." );
      System.exit(0);
   }

   /* Iterate through the lines in the list. */
   while ( filtlist.hasMoreElements() ) {
      LDAPFilter filtline = filtlist.next();

      /* Add the prefix and suffix, and generate the filter. */
```

```
      String prefix = "(&(l=Sunnyvale)";
      String suffix = ")";
      filtline.setFilterAffixes( prefix, suffix );
      String filterString = filtline.getFilter( searchTerm );
      System.out.println( "  Generated filter string: " + filterString +
         "\n" );
   }
} catch ( Exception e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

For example, if the following search term is passed to the `LDAPFilterDescriptor.getFilters` method:

```
bjensen@airius.com
```

and the corresponding filter (without a prefix or suffix) is:

```
(mail=bjensen@airius.com)
```

the entire generated filter string (retrieved by the `LDAPFilter.getFilter` method) will be:

```
(&(l=Sunnyvale)(mail=bjensen@airius.com))
```

Adding Affixes By Using getFilter

The following section of code loads the filter configuration file named `myfilters.conf` into memory and uses the `LDAPFilter.getFilter` method to add a prefix and suffix to a generated filter:

```
...
import netscape.ldap.util.*;
...
LDAPFilterDescriptor filtdesc = null;
try {
   /* Read in the filter configuration file. */
   filtdesc = new LDAPFilterDescriptor( "myfilter.conf" );

   /* Get filters from the section "people" for the search term. */
   LDAPFilterList filtlist = null;
   try {
      filtlist = filtdesc.getFilters( "people", searchTerm );
   } catch ( Exception e ) {
      System.out.println( "No matching tag section or filter found." );
      System.exit(0);
   }
```

```
   /* Iterate through the lines in the list. */
   while ( filtlist.hasMoreElements() ) {
      LDAPFilter filtline = filtlist.next();

      /* Add the prefix and suffix, and generate the filter. */
      String prefix = "(&(l=Sunnyvale)";
      String suffix = ")";
      String filterString = filtline.getFilter( searchTerm, prefix,
         suffix );
      System.out.println( "  Generated filter string: " + filterString +
         "\n" );
   }
} catch ( Exception e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

For example, if the following search term is passed to the `LDAPFilterDescriptor.getFilters` method:

```
bjensen@airius.com
```

and the corresponding filter (without a prefix or suffix) is:

```
(mail=bjensen@airius.com)
```

the entire generated filter string (retrieved by the `LDAPFilter.getFilter` method) will be:

```
(&(l=Sunnyvale)(mail=bjensen@airius.com))
```

Adding Affixes By Using setupFilter

The following section of code loads the filter configuration file named `myfilters.conf` into memory and uses the `LDAPFilter.setupFilter` method to add a prefix and suffix to a generated filter:

```
...
import netscape.ldap.util.*;
...
LDAPFilterDescriptor filtdesc = null;
try {
   /* Read in the filter configuration file. */
   filtdesc = new LDAPFilterDescriptor( "myfilter.conf" );

   /* Get filters from the section "people" for the search term. */
   LDAPFilterList filtlist = null;
```

```
   try {
      filtlist = filtdesc.getFilters( "people", searchTerm );
   } catch ( Exception e ) {
      System.out.println( "No matching tag section or filter found." );
      System.exit(0);
   }

   /* Iterate through the lines in the list. */
   while ( filtlist.hasMoreElements() ) {
      LDAPFilter filtline = filtlist.next();

      /* Add the prefix and suffix, and generate the filter. */
      String prefix = "(&(l=Sunnyvale)";
      String suffix = ")";
      filtline.setupFilter( searchTerm, prefix, suffix );

      String filterString = filtline.getFilter();System.out.println( "
Generated filter string: " + filterString +
         "\n" );
   }
} catch ( Exception e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

For example, if the following search term is passed to the `LDAPFilterDescriptor.getFilters` method:

```
bjensen@airius.com
```

and the corresponding filter (without a prefix or suffix) is:

```
(mail=bjensen@airius.com)
```

the entire generated filter string (retrieved by the `LDAPFilter.getFilter` method) will be:

```
(&(l=Sunnyvale)(mail=bjensen@airius.com))
```

Chapter

8

Adding, Updating, and Deleting Entries

This chapter explains how to use the LDAP Java classes to add, modify, delete, and rename entries in the directory.

The chapter includes the following sections:

- “Adding a New Entry”
- “Modifying an Entry”
- “Deleting an Entry”
- “Changing the Name of an Entry”

Adding a New Entry

To add an entry to the directory, you need to follow this general procedure:

1. Create each individual attribute that will be in the entry. (See “Creating a New Attribute”.)
2. Create the set of attributes that make up the entry, and add each of the attributes to this set. (See “Creating a New Attribute Set”.)

3. Create the new entry, specifying a unique distinguished name (DN) to identify the entry and the set of attributes that make up the entry. (See "Creating a New Entry".)

4. Add the new entry to directory. (See "Adding the Entry to the Directory".)

For a complete example, see "Example of Adding an Entry".

Creating a New Attribute

An attribute can have a single value or multiple values and can contain string values or binary data. In the LDAP Java classes, an attribute is represented by an `LDAPAttribute` object.

To create a new attribute, use the `LDAPAttribute` constructor. You can specify a single string value, multiple string values, or a binary value when constructing the object.

For example, the following section of code creates a new object for the attribute "cn" with the value "Jane St. Clair".

```
...
LDAPAttribute attr = new LDAPAttribute( "cn", "Jane St. Clair" );
...
```

The following section of code creates an attribute "objectclass" with the values "top", "person", "organizationalPerson", and "inetOrgPerson".

```
...
String objectclasses[] = { "top",
   "person", "organizationalPerson", "inetOrgPerson" };
LDAPAttribute attr = new LDAPAttribute( "objectclass", objectclasses );
...
```

You can also add string or binary values to an `LDAPAttribute` object by invoking the `addValue` method.

Creating a New Attribute Set

To specify the set of attributes in an entry, you need to create an attribute set. In the LDAP Java classes, a set of one or more attributes is represented by an `LDAPAttributeSet` object.

To create a new attribute set, use the `LDAPAttributeSet` constructor and invoke the `add` method to add `LDAPAttribute` objects to the set.

For example:

```
...
LDAPAttribute attr1 = new LDAPAttribute( "cn", "Jane St. Clair" );
String objectclasses[] = { "top",
   "person", "organizationalPerson", "inetOrgPerson" };
LDAPAttribute attr2 = new LDAPAttribute( "objectclass", objectclasses );
LDAPAttributeSet attrSet = new LDAPAttributeSet();
attrSet.add( attr1 );
attrSet.add( attr2 );
...
```

Creating a New Entry

An entry contains a distinguished name (DN), which identifies the entry in the directory, and a set of attributes. In the LDAP Java classes, an entry is represented by an `LDAPEntry` object.

To create a new entry, use the `LDAPEntry` constructor. For example:

```
...
LDAPAttribute attr1 = new LDAPAttribute( "cn", "Jane St. Clair" );
String objectclasses[] = { "top",
   "person", "organizationalPerson", "inetOrgPerson" };
LDAPAttribute attr2 = new LDAPAttribute( "objectclass", objectclasses );
LDAPAttributeSet attrSet = new LDAPAttributeSet();
attrSet.add( attr1 );
attrSet.add( attr2 );
String dn = "uid=jsclair,ou=People,o=Airius.com";
LDAPEntry newEntry = new LDAPEntry( dn, attrs );
...
```

Adding the Entry to the Directory

To add the entry to the directory, invoke the `add` method of the `LDAPConnection` object. For example:

```
...
try {
   LDAPConnection ld = new LDAPConnection();
   ld.connect( "localhost", LDAPv2.DEFAULT_PORT );
   ld.authenticate( "cn=Directory Manager", "23skidoo" );
```

```
   LDAPEntry newEntry = new LDAPEntry( dn, attrs );
   ld.add( newEntry );
...
```

Before you add an entry, make sure of the following:

- You have specified the object classes of the entry (use the “objectclass” attribute to specify these) and the required attributes for that object class.

  For example, in the Netscape Directory Server, organizational units are represented by entries of the “organizationalUnit” object class. To add an entry for a person, you need to specify the following attributes in the entry:

  - objectclass (this attribute should have the values “top” and “organizationalUnit”)
  - ou (this is a required attribute)

  For a listing of object classes and their required attributes, see the Directory Server Administrator’s Guide.

- Make sure that you authenticate as a user who has the access permissions to add the entry to the directory. (If you do not have permission to add the entry, an `LDAPException` is thrown with the result code `LDAPException.INSUFFICIENT_ACCESS_RIGHTS`.)

Example of Adding an Entry

The following example adds a new entry to the directory for the user named William Jensen.

```
import netscape.ldap.*;
import java.util.*;
public class Add {
   public static void main( String[] args ) {
      /* Specify the DN of the new entry. */
      String dn = "uid=wbjensen, ou=People, o=Airius.com";

      /* Create a new attribute set for the entry. */
      LDAPAttributeSet attrs = new LDAPAttributeSet();

      /* Create and add attributes to the attribute set. */
      String objectclass_values[] = { "top", "person",
         "organizationalPerson", "inetOrgPerson" };
      LDAPAttribute attr = new LDAPAttribute( "objectclass",
         objectclass_values );
```

```
attrs.add( attr );

String cn_values[] = { "William B Jensen", "William Jensen",
   "Bill Jensen" };
attr = new LDAPAttribute( "cn", cn_values );
attrs.add( attr );

String givenname_values[] = { "William", "Bill" };
attr = new LDAPAttribute( "givenname", givenname_values );
attrs.add( attr );

attrs.add( new LDAPAttribute( "sn", "Jensen" ) );
attrs.add( new LDAPAttribute( "telephonenumber",
   "+1 415 555 1212" ) );
attrs.add( new LDAPAttribute( "uid", "wbjensen" ) );

/* Create an entry with this DN and these attributes . */
LDAPEntry myEntry = new LDAPEntry( dn, attrs );

/* Connect to the server and add the entry. */
LDAPConnection ld = null;
int status = -1;
try {
   ld = new LDAPConnection();

   /* Connect to the server. */
   String HOSTNAME= "localhost";
   ld.connect( HOSTNAME, LDAPv2.DEFAULT_PORT );

   /* Authenticate to the server as the directory manager. */
   String MGR_DN = "cn=Directory Manager";
   String MGR_PW = "23skidoo";
   ld.authenticate( MGR_DN, MGR_PW );

   /* Add the entry to the directory. */
   ld.add( myEntry );
   System.out.println( "Added entry successfully."  );

}
catch( LDAPException e ) {
   if (e.getLDAPResultCode() ==
      LDAPException.ENTRY_ALREADY_EXISTS )
      System.out.println( "Error: Entry already present" );
   else
      System.out.println( "Error: " + e.toString() );
}

/* When done, disconnect from the server. */
if ( (ld != null) && ld.isConnected() ) {
   try {
```

```
            ld.disconnect();
         } catch ( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
         }
      }
      System.exit(status);
   }
}
```

Modifying an Entry

To modify an entry in the directory, you need to follow this general procedure:

1. Specify each change to an attribute that needs to be made. You can do one of the following:

   - If you are making only one change to the entry, you need to construct an `LDAPModification` object specifying the change that needs to be made.
   - If you are making more than one change, you need to construct an `LDAPModificationSet` object specifying the changes that need to be made.

   See “Specifying the Changes” for details.

2. Use the distinguished name of the entry to find and update the entry in the directory. (See “Modifying the Entry in the Directory”.)

For a complete example, see “Example of Modifying an Entry”.

Specifying the Changes

You can add a new attribute, removing an existing attribute, or changing the values of an existing attribute. This section describes the process of specifying these changes.

- Adding New Values to an Attribute
- Removing Values to an Attribute

- Replacing the Values of an Attribute
- Adding a New Attribute
- Removing an Attribute

Adding New Values to an Attribute

To add new values to an attribute in an entry, construct a new `LDAPAttribute` object, specifying the name of the attribute and the values that you want to add. Then, do one of the following:

- If you are only making a single change to the entry, construct a new `LDAPModification` object to specify that change. Pass `LDAPModification.ADD` and the `LDAPAttribute` object as arguments to the `LDAPModification` constructor.
- If you are collecting multiple changes to an entry in an `LDAPModificationSet` object, invoke the `add` method to add this change to the list. Pass `LDAPModification.ADD` and the `LDAPAttribute` object as arguments to this method.

For example, the following section of code prepares to add the value “babs@airius.com” to the “mail” attribute to an entry.

```
...
LDAPModificationSet mods = new LDAPModificationSet();
LDAPAttribute attrMail = new LDAPAttribute( "mail",
   "babs@airius.com" );
mods.add( LDAPModification.ADD, attrMail );
...
```

Note that if the specified attribute does not exist in the entry, the attribute will be added to the entry. For example, if the “mail” attribute does not exist in the entry, adding the value “babs@airius.com” will add the “mail” attribute to the entry.

Removing Values to an Attribute

To remove values from an attribute in an entry, construct a new `LDAPAttribute` object, specifying the name of the attribute and the values that you want to remove. Then, do one of the following:

- If you are only making a single change to the entry, construct a new `LDAPModification` object to specify that change. Pass `LDAPModification.DELETE` and the `LDAPAttribute` object as arguments to the `LDAPModification` constructor.
- If you are collecting multiple changes to an entry in an `LDAPModificationSet` object, invoke the `add` method to add this change to the list. Pass `LDAPModification.DELETE` and the `LDAPAttribute` object as arguments to this method.

For example, the following section of code prepares to remove the value "babs@airius.com" from the "mail" attribute to an entry.

```
...
LDAPModificationSet mods = new LDAPModificationSet();
LDAPAttribute attrMail = new LDAPAttribute( "mail",
   "babs@airius.com" );
mods.add( LDAPModification.DELETE, attrMail );
...
```

Note that if you are removing all values in the attribute, the attribute will be removed. For example, if "babs@airius.com" is the only value of the "mail" attribute, removing this value will remove the "mail" attribute from the entry.

Also note that if you do not specify any values in the `LDAPAttribute` object, the attribute will be removed. For example, if you construct the object by invoking `LDAPAttribute( "mail" )`, the "mail" attribute will be removed.

Replacing the Values of an Attribute

To replace all of the values of an attribute in an entry, construct a new `LDAPAttribute` object, specifying the name of the attribute and the new values that should replace all of the existing values of the attribute. Then, do one of the following:

- If you are only making a single change to the entry, construct a new `LDAPModification` object to specify that change. Pass `LDAPModification.REPLACE` and the `LDAPAttribute` object as arguments to the `LDAPModification` constructor.
- If you are collecting multiple changes to an entry in an `LDAPModificationSet` object, invoke the `add` method to add this change to the list. Pass `LDAPModification.REPLACE` and the `LDAPAttribute` object as arguments to this method.

For example, the following section of code prepares to replace the existing values of the “mail” attribute with the values “bjensen@airius.com” and “babs@airius.com”.

```
...
LDAPModificationSet mods = new LDAPModificationSet();
String attrValues = { "bjensen@airius.com", "babs@airius.com" }
LDAPAttribute attrMail = new LDAPAttribute( "mail", attrValues );
mods.add( LDAPModification.REPLACE, attrMail );
...
```

Note that if the specified attribute does not exist in the entry, the attribute will be added to the entry. For example, if the “mail” attribute does not exist in the entry, replacing the values “bjensen@airius.com” and “babs@airius.com” will add the “mail” attribute to the entry.

Also note that if you do not specify any values in the `LDAPAttribute` object, the attribute will be removed. For example, if you construct the object by invoking `LDAPAttribute( "mail" )`, the “mail” attribute will be removed.

Adding a New Attribute

To add a new attribute to an entry, follow the instructions under “Adding New Values to an Attribute” or “Replacing the Values of an Attribute”. If you add or replace values for an attribute that does not exist in the entry, the attribute will be added to the entry.

Removing an Attribute

To remove an attribute from an entry, you can do one of the following:

- replace the values of the attribute with no values (construct the `LDAPAttribute` object with no values)
- specify that you want to remove a value from the attribute, and specify no value (construct the `LDAPAttribute` object with no values)
- remove all values of the attribute

For example, the following section of code prepares to remove the “mail” and “description” attributes from an entry. This example demonstrates how you can use either `LDAPModification.REPLACE` or `LDAPModification.DELETE` to remove the attribute.

```
...
LDAPModificationSet mods = new LDAPModificationSet();
LDAPAttribute attrMail = new LDAPAttribute( "mail" );
LDAPAttribute attrDesc = new LDAPAttribute( "description" );
mods.add( LDAPModification.REPLACE, attrMail );
mods.add( LDAPModification.DELETE, attrDesc );
...
```

For more information on removing or replacing values, see “Removing Values to an Attribute” or “Replacing the Values of an Attribute”.

Modifying the Entry in the Directory

When you are done specifying the change (an `LDAPModification` object) or list of changes (`LDAPModificationSet`) that you want made, you can pass this object with the distinguished name of the entry that you want modified to the `modify` method of the `LDAPConnection` object.

Before you modify an entry, make sure of the following:

- You are not removing any of the required attributes for that object class.

  For example, in the Netscape Directory Server, organizational units are represented by entries of the “organizationalUnit” object class. The “ou” attribute is a required attribute of the object class and should not be removed.

  For a listing of object classes and their required attributes, see the Directory Server Administrator’s Guide.

- Make sure that you authenticate as a user who has the access permissions to modify the entry in the directory. (If you do not have permission to modify the entry, an `LDAPException` is thrown with the result code `LDAPException.INSUFFICIENT_ACCESS_RIGHTS`.)

Example of Modifying an Entry

The following example modifies the entry in the directory for the user named William Jensen.

```
import netscape.ldap.*;
import java.util.*;
public class ModAttrs {
```

```
public static void main( String[] args ) {
   /* Specify the entry to be modified. */
   String ENTRYDN = "uid=wbjensen, ou=People, o=Airius.com";

   /* Create a new set of modifications. */
   LDAPModificationSet mods = new LDAPModificationSet();

   /* Add each change to an attribute to the set of modifications. */
   LDAPAttribute attrEmail = new LDAPAttribute( "mail",
      "willj@airius.com" );
   mods.add( LDAPModification.REPLACE, attrEmail );

   LDAPAttribute attrDesc = new LDAPAttribute( "description",
      "This entry was modified with the modattrs program" );
   mods.add( LDAPModification.ADD, attrDesc );

   LDAPAttribute attrPhone = new
      LDAPAttribute("telephoneNumber");
   mods.add( LDAPModification.DELETE, attrPhone );

   /* Connect to the server and modify the entry. */
   LDAPConnection ld = null;
   int status = -1;
   try {
      ld = new LDAPConnection();

      /* Connect to the server. */
      String HOSTNAME = "localhost";
      ld.connect( HOSTNAME, LDAPv2.DEFAULT_PORT );

      /* Authenticate to the server as directory manager */
      String MGR_DN = "cn=Directory Manager";
      String MGR_PW = "23skidoo";
      ld.authenticate( MGR_DN, MGR_PW );

      /* Now modify the entry in the directory */
      ld.modify( ENTRYDN, mods );
      System.out.println( "Successfully modified the entry." );
   } catch( LDAPException e ) {
      if ( e.getLDAPResultCode() ==
         LDAPException.NO_SUCH_OBJECT )
         System.out.println( "Error: No such entry" );
      else if ( e.getLDAPResultCode() ==
         LDAPException.INSUFFICIENT_ACCESS_RIGHTS )
         System.out.println( "Error: Insufficient rights" );
      else if ( e.getLDAPResultCode() ==
         LDAPException.ATTRIBUTE_OR_VALUE_EXISTS )
         System.out.println( "Error: Attribute or value exists" );
      else
         System.out.println( "Error: " + e.toString() );
```

```
      }

      /* Disconnect when done. */
      if ( (ld != null) && ld.isConnected() ) {
         try {
            ld.disconnect();
         } catch ( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
         }
      }
      System.exit(status);
   }
}
```

Deleting an Entry

To remove an entry from the directory, invoke the `delete` method of the `LDAPConnection` object and specify the distinguished name (DN) of the entry that you want to remove.

Before you delete an entry, make sure that you authenticate as a user who has the access permissions to remove the entry from the directory. (If you do not have permission to remove the entry, an `LDAPException` is thrown with the result code `LDAPException.INSUFFICIENT_ACCESS_RIGHTS`.)

Example of Deleting an Entry

The following section of code deletes the entry for the user named William Jensen.

```
import netscape.ldap.*;
import java.util.*;
public class Del {
   public static void main( String[] args ) {

      /* Connect to the server and delete the entry. */
      LDAPConnection ld = null;
      int status = -1;
      try {
         ld = new LDAPConnection();

         /* Connect to the server. */
         String HOSTNAME = "localhost";
```

```
            ld.connect( HOSTNAME, LDAPv2.DEFAULT_PORT );

            /* Authenticate to the server as the directory manager. */
            String MGR_DN = "cn=Directory Manager";
            String MGR_PW = "23skidoo";
            ld.authenticate( MGR_DN, MGR_PW );

            /* Delete the entry. */
            String dn = "uid=wbjensen, ou=People, o=Airius.com";
            ld.delete( dn );
            System.out.println( "Entry deleted" );
         }

         catch( LDAPException e ) {
            if ( e.getLDAPResultCode() ==
               LDAPException.NO_SUCH_OBJECT )
               System.out.println( "Error: No such entry" );
            else if ( e.getLDAPResultCode() ==
               LDAPException.INSUFFICIENT_ACCESS_RIGHTS )
               System.out.println( "Error: Insufficient rights" );
            else
               System.out.println( "Error: " + e.toString() );
         }
         /* When done, disconnect from the server. */
         if ( (ld != null) && ld.isConnected() ) {
            try {
               ld.disconnect();
            } catch ( LDAPException e ) {
               System.out.println( "Error: " + e.toString() );
            }
         }
         System.exit(status);
      }
   }
```

Changing the Name of an Entry

To rename an entry, invoke the `rename` method of the `LDAPConnection` object. Using this method, you can do the following:

- Change the relative distinguished name (RDN) of an entry. For example, you can
- Change the location of an entry in the directory tree (in other words, change the distinguished name of an entry)

Note that not all LDAP servers support the ability to change the location of an entry in the directory tree. (The Netscape Directory Server does not yet support this capability.)

For a complete example, see “Example of Renaming an Entry”.

Removing the Attribute for the Old RDN

When invoking the `rename` method of the `LDAPConnection` object, you can specify a `deleteoldrdn` parameter that allows you to remove the old RDN from the entry. The `deleteoldrdn` parameter is best explained through this example. Suppose an entry has the following values for the `uid` attribute:

```
uid: wbjensen
uid: wbj
```

The following method adds "wjensen" to this list of values and removes the "wbjensen" value:

```
ld.rename( "uid=wbjensen,ou=People,o=Airius.com", "uid=wjensen",
   true );
```

The resulting entry has the following values:

```
uid: wjensen
uid: wbj
```

If instead `false` is passed for the `deleteoldrdn` parameter:

```
ld.rename( "uid=wbjensen,ou=People,o=Airius.com", "uid=wjensen",
   false );
```

the "Barbara Jensen" value is not removed from the entry:

```
uid: wjensen
uid: wbjensen
uid: wbj
```

Before you rename an entry, make sure that you authenticate as a user who has the access permissions to rename the entry in the directory. (If you do not have permission to rename the entry, an `LDAPException` is thrown with the result code `LDAPException.INSUFFICIENT_ACCESS_RIGHTS`.)

Example of Renaming an Entry

The following example changes the RDN of an entry from uid=wbjensen to uid=wjensen.

```
import netscape.ldap.*;
import java.util.*;
public class Rename {
   public static void main( String[] args ) {
      /* Connect to the server and rename the entry. */
      LDAPConnection ld = null;
      int status = -1;
      try {
         ld = new LDAPConnection();

         /* Connect to the server. */
         String HOSTNAME = "localhost";
         ld.connect( HOSTNAME, LDAPv2.DEFAULT_PORT );

         /* Authenticate to the server as the directory manager. */
         String MGR_DN = "cn=Directory Manager";
         String MGR_PW = "23skidoo";
         ld.authenticate( MGR_DN, MGR_PW );

         /* Change the RDN of the entry. */
         String dn = "uid=wbjensen,ou=People,o=Airius.com";
         String nrdn = "uid=wjensen";
         ld.rename( dn, nrdn, true );
         System.out.println( "Entry " + dn + " has been renamed." );
      }
      catch( LDAPException e ) {
         if ( e.getLDAPResultCode() ==
            LDAPException.NO_SUCH_OBJECT )
            System.out.println( "Error: No such entry" );
         else if ( e.getLDAPResultCode() ==
            LDAPException.INSUFFICIENT_ACCESS_RIGHTS )
            System.out.println( "Error: Insufficient rights" );
         else if ( e.getLDAPResultCode() ==
            LDAPException.ATTRIBUTE_OR_VALUE_EXISTS )
            System.out.println( "Error: Attribute or value exists" );
         else
            System.out.println( "Error: " + e.toString() );
      }
      /* Done, so disconnect */
      if ( (ld != null) && ld.isConnected() ) {
         try {
            ld.disconnect();
         } catch ( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
```

```
            }
        }
        System.exit(status);
    }
}
```

Chapter

9

Comparing Values in Entries

This chapter explains how to compare the value of an attribute in an entry against a specified value.

Comparing the Value of an Attribute

The LDAP Java classes allow you to compare a specified value against the value of an entry in the directory. For example, you can check to see if the "mail" attribute of an entry contains the value "bjensen@airius.com".

To compare a specified value against an attribute of an entry in the directory, you need to follow this general procedure:

1. Specify the name of the attribute that you want to check and the value that you want to use for comparison. (See "Specifying the Attribute and Value".)
2. Use the distinguished name of the entry to find the entry in the directory and perform the comparison. (See "Performing the Comparison".)

For a complete example, see "Example of Comparing a Value Against an Attribute".

Specifying the Attribute and Value

Use an `LDAPAttribute` object to specify the name of the attribute that you want to check and the value that you want to use in the comparison.

For example, to compare the value “bjensen@airius.com” against the value of the “mail” attribute in an entry, use the following constructor:

```
...
LDAPAttribute attr = new LDAPAttribute( "mail", "bjensen@airius.com" );
...
```

Performing the Comparison

To perform the comparison, use the `compare` method of the `LDAPConnection` object. Specify the distinguished name (DN) of the entry that you want to compare.

This method returns `true` if the attribute in the entry contains the specified value.

Example of Comparing a Value Against an Attribute

The following section of code determines if the values “person” and “xyzzy” are in the “objectclass” attribute of the entry for Barbara Jensen.

```
import netscape.ldap.*;
import java.util.*;
public class Compare {
   public static void main( String[] args ) {

      /* Connect to the server and perform the comparison. */
      LDAPConnection ld = null;
      int status = -1;
      try {
         ld = new LDAPConnection();

         /* Connect to the server. */
         String HOSTNAME = "localhost";
         ld.connect( HOSTNAME, LDAPv2.DEFAULT_PORT );
```

```
            /* Authenticate to the server as the directory manager. */
            String MGR_DN = "cn=Directory Manager";
            String MGR_PW = "23skidoo";
            ld.authenticate( MGR_DN, MGR_PW );

            /* Perform the comparisons. */
            String ENTRYDN = "uid=bjensen, ou=People, o=Airius.com";
            LDAPAttribute attr = new LDAPAttribute( "objectclass",
               "person" );
            boolean ok = ld.compare( ENTRYDN, attr );
            reportResults( ok, attr );

            attr = new LDAPAttribute( "objectclass", "xyzzy" );
            ok = ld.compare( ENTRYDN, attr );
            reportResults( ok, attr );
         }
         catch( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
         }

         /* Done, so disconnect */
         if ( (ld != null) && ld.isConnected() ) {
            try {
               ld.disconnect();
            } catch ( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
            }
         }
         System.exit(status);
      }

      /* Print the results of the comparison. */
      private static void reportResults( boolean ok, LDAPAttribute attr ) {
         String result;
         if ( ok )
            result = new String();
         else
            result = new String( "not " );
         Enumeration en = attr.getStringValues();
         if (en != null) {
            String val = (String)en.nextElement();
            System.out.println(
               "The value " + val + " is " + result + "contained in the " +
               attr.getName() + " attribute." );
         }
      }
   }
```

Chapter

10

Working with LDAP URLs

This chapter describes what LDAP URLs are and explains how to use LDAP URLs to search and retrieve data from the directory.

The chapter contains the following sections:

- "Understanding LDAP URLs"
- "Examples of LDAP URLs"
- "Getting the Components of an LDAP URL"
- "Processing an LDAP URL"

Understanding LDAP URLs

An LDAP URL is a URL that begins with the `ldap://` protocol prefix (or `ldaps://`, if the server is communicating over an SSL connection) and specifies a search request to be sent to an LDAP server.

In the LDAP Java classes, you can represent an LDAP URL as an `LDAPUrl` object. You can invoke methods of this object to parse an LDAP URL into its components and to process a search request specified by an LDAP URL.

LDAP URLs have the following syntax:

```
ldap[s]://<hostname>:<port>/<base_dn>?<attributes>?<scope>?<filter>
```

(RFC 2255, The LDAP URL Format, also specifies that a "bindname" extension can be present at the end of the URL. At this point in time, the LDAP Java classes do not handle this extension.)

The `ldap://` protocol is used to connect to LDAP servers over unsecured connections, and the `ldaps://` protocol is used to connect to LDAP servers over SSL connections.

Table 10.1 lists the components of an LDAP URL.

Table 10.1 Components of an LDAP URL

Component	Description
<hostname>	Name (or IP address in dotted format) of the LDAP server (for example, `ldap.netscape.com` or `192.202.185.90`).
<port>	Port number of the LDAP server (for example, 696). If no port is specified, the standard LDAP port (389) is used.
<base_dn>	Distinguished name (DN) of an entry in the directory. This DN identifies the entry that is starting point of the search. If this component is empty, the search starts at the root DN.
<attributes>	The attributes to be returned. To specify more than one attribute, use commas to delimit the attributes (for example, "cn,mail,telephoneNumber"). If no attributes are specified in the URL, all attributes are returned.

Table 10.1 Components of an LDAP URL

Component	Description
<scope>	The scope of the search, which can be one of these values: • `base` retrieves information only about the distinguished name (<*base_dn*>) specified in the URL. • `one` retrieves information about entries one level below the distinguished name (<*base_dn*>) specified in the URL. The base entry is not included in this scope. • `sub` retrieves information about entries at all levels below the distinguished name (<*base_dn*>) specified in the URL. The base entry is included in this scope. If no scope is specified, the server performs a `base` search.
<filter>	Search filter to apply to entries within the specified scope of the search. If no filter is specified, the server uses the filter `(objectClass=*)`.

Any "unsafe" characters in the URL need to be represented by a special sequence of characters (this is often called escaping unsafe characters). For example, a space must be represented as `%20`. Thus, the distinguished name `"ou=Product Development"` must be encoded as `"ou=Product%20Development"`.

Note that *<attributes>*, *<scope>*, and *<filter>* are identified by their positions in the URL. If you do not want to specify any attributes, you still need to include the question marks delimiting that field.

For example, to specify a subtree search starting from `"o=Airius.com"` that returns all attributes for entries matching `"(sn=Jensen)"`, use the following URL:

```
ldap://ldap.netscape.com/o=Airius.com??sub?(sn=Jensen)
```

Note that the two consecutive question marks — ?? — indicate that no attributes have been specified. Since no specific attributes are identified in the URL, all attributes are returned in the search.

Examples of LDAP URLs

The following LDAP URL specifies a base search for the entry with the distinguished name "o=Airius.com".

```
ldap://ldap.netscape.com/o=Airius.com
```

- Because no port number is specified, the standard LDAP port number (389) is used.
- Because no attributes are specified, the search returns all attributes.
- Because no search scope is specified, the search is restricted to the base entry "o=Airius.com".
- Because no filter is specified, the default filter "(objectclass=*)" is used.

The following LDAP URL retrieves the postalAddress attribute of the o=Airius.com entry:

```
ldap://ldap.netscape.com/o=Airius.com?postalAddress
```

- Because no search scope is specified, the search is restricted to the base entry "o=Airius.com".
- Because no filter is specified, the default filter "(objectclass=*)" is used.

The following LDAP URL retrieves the cn, mail, and telephoneNumber attributes of the entry for Barbara Jensen:

```
ldap://ldap.netscape.com/
uid=bjensen,ou=People,o=Airius.com?cn,mail,telephoneNumber
```

- Because no search scope is specified, the search is restricted to the base entry "uid=bjensen,ou=People,o=Airius.com".
- Because no filter is specified, the default filter "(objectclass=*)" is used.

The following LDAP URL specifies a search for entries that have the last name Jensen and are at any level under "o=Airius.com":

```
ldap://ldap.netscape.com/o=Airius.com??sub?(sn=Jensen)
```

- Because no attributes are specified, the search returns all attributes.
- Because the search scope is sub, the search encompasses the base entry "o=Airius.com" and entries at all levels under the base entry.

The following LDAP URL specifies a search for the object class for all entries one level under `"o=Airius.com"`:

```
ldap://ldap.netscape.com/o=Airius.com?objectClass?one
```

- Because the search scope is `one`, the search encompasses all entries one level under the base entry `"o=Airius.com"`. The search scope does not include the base entry.
- Because no filter is specified, the default filter `"(objectclass=*)"` is used.

Important The syntax for LDAP URLs does not include any means for specifying credentials or passwords. Search requests initiated through LDAP URLs are unauthenticated.

Getting the Components of an LDAP URL

To get the individual components of an LDAP URL, pass the URL to the `LDAPUrl` constructor to create a new `LDAPUrl` object, then use the following methods:

- To get an array of the attributes that should be returned in the search results, use the `getAttributeArray` method. To get these attributes as an enumeration, use the `getAttributes` method.
- To get the hostname of the LDAP server, use the `getHost` method.
- To get the port number of the LDAP server, use the `getPort` method.
- To get the base DN, use the `getDN` method.
- To get the scope of the search, use the `getScope` method.
- To get the search filter, use the `getFilter` method.

Processing an LDAP URL

To process the search request specified by an LDAP URL, you can invoke one of the following methods, passing in the `LDAPUrl` object:

- If the URL specifies a base search for a single entry, invoke the `read` method of the `LDAPConnection` object to read the entry from the directory.
- Otherwise, invoke the `search` method of the `LDAPConnection` object to perform the search.

Both methods create a new `LDAPConnection` object, connect to the LDAP server specified in the URL, perform the search, and disconnect.

3

Advanced Topics

Chapter 11 Getting Server Information

This chapter explains how to access and modify information about your LDAP server over the LDAP protocol.

Chapter 12 Connecting Over SSL

This chapter describes the process of enabling an LDAP client to connect to an LDAP server over the Secure Sockets Layer (SSL) protocol. The chapter covers the procedures for connecting to an LDAP server and authenticating.

Chapter 13 Working with LDAP Controls

This chapter explains how LDAP controls work and how to use the LDAP controls that are supported by the Netscape Directory Server.

Chapter 14 Using SASL Authentication

This chapter describes the process of using a SASL mechanism to authenticate an LDAP client to an LDAP server.

Chapter 15 Using the JNDI Service Provider

This chapter explains JNDI and shows you how to use Netscape's LDAP Service Provider for JNDI.

Chapter 16 Working with Extended Operations

This chapter explains how LDAP v3 extended operations work and how to use the extended operations that are supported by your LDAP server.

Chapter 17 Using the Asynchronous Interface

This chapter shows you how to use the Asynchronous Interface to LDAP in Java applications.

Chapter

11

Getting Server Information

This chapter explains how to access and modify information about your LDAP server over the LDAP protocol.

The chapter includes the following sections:

- “Understanding DSEs”
- “Getting the Root DSE”
- “Determining If the Server Supports LDAP v3”
- “Getting Schema Information”

Understanding DSEs

A DSE is a DSA-specific entry in the directory. (A DSA is a directory system agent, which is an X.500 term for a directory server.) A DSE contains information specific to the server.

In a directory tree, the root of the tree is the root DSE. It is not part of any naming context (for example, it is above "o=Airius.com" in the directory tree).

(Note that the root DSE is specified as part of the LDAP v3 protocol. LDAP v2 servers do not necessarily have a root DSE.)

The root DSE can contain the following information:

- the naming contexts of this server (for example, "o=Airius.com")
- URLs of alternate servers to contact if this server is unavailable
- the LDAP v3 extended operations supported by this server (see Chapter 16, “Working with Extended Operations”for details)
- the LDAP v3 controls supported by this server (see Chapter 13, “Working with LDAP Controls” for details)
- the SASL mechanisms supported by this server (see Chapter 14, “Using SASL Authentication” for details)
- the versions of the LDAP protocol supported by this server (for example, 2 and 3)
- additional server-specific information

Getting the Root DSE

The root DSE for an LDAP server specifies information about the server. The following table lists the types of information available in different attributes of the root DSE.

Table 11.1 Information available in the root DSE

Attribute Name	Description of Values
namingContexts	The values of this attribute are the naming contexts supported by this server (for example, "o=Airius.com").
altServer	The values of this attribute are LDAP URLs that identify other servers that can be contacted if this server is unavailable.
supportedExtension	The values of this attribute are the object identifiers (OIDs) of the LDAP v3 extended operations supported by this server. If this attribute is not in the root DSE, the server does not support any extended operations.

Table 11.1 Information available in the root DSE

Attribute Name	Description of Values
supportedControl	The values of this attribute are the object identifiers (OIDs) of the LDAP v3 controls supported by this server. If this attribute is not in the root DSE, the server does not support any LDAP v3 controls.
supportedSASLMechanisms	The values of this attribute are the names of the SASL mechanisms supported by the server. If this attribute is not in the root DSE, the server does not support any SASL mechanisms.
supportedLDAPVersion	The values of this attribute are the versions of the LDAP protocol supported by this server (for example, 2 and 3).

To get the root DSE for an LDAP server, do the following:

1. Turn off automatic referral handling and connect to the LDAP server (see "Creating a Connection and Setting Preferences", "Connecting to the LDAP Server", and "Enabling or Disabling Referral Handling" for details).
2. Search the directory using the following criteria:
   - Set the search scope to a base search.
   - Specify an empty string for the base DN.
   - Use the search filter `(objectclass=*)`.

   For details on how to use Java classes to search the directory, see Chapter 6, "Searching the Directory".

   If an `LDAPException` is thrown with a result code such as `OPERATION_ERROR`, `PROTOCOL_ERROR`, `REFERRAL`, or `NO_SUCH_OBJECT`, the LDAP server probably does not support LDAP v3.

The following section of code gets the root DSE for a server and prints out its attributes.

```
...
import netscape.ldap.*;
import java.util.*;
...
```

```
/* Create a new connection. */
LDAPConnection ld = new LDAPConnection();
String hostname = "localhost";
int portnumber = LDAPv2.DEFAULT_PORT;

try {
   /* Connect to the LDAP server. */
   ld.connect( 3, hostname, portnumber );

   /* Get the root DSE by doing a search where:
         - The scope is SCOPE_BASE
         - The base is ""
         - The search filter is "(objectclass=*)"
   */
   int MY_SCOPE = LDAPv2.SCOPE_BASE;
   String MY_FILTER = "(objectclass=*)";
   String MY_SEARCHBASE = "";
   LDAPSearchResults res = ld.search( MY_SEARCHBASE,
      MY_SCOPE, MY_FILTER, null, false );

   /* There should be only one entry in the results (the root DSE). */
   while ( res.hasMoreElements() ) {
      LDAPEntry findEntry = (LDAPEntry)res.nextElement();

      /* Get the attributes of the root DSE. */
      LDAPAttributeSet findAttrs = findEntry.getAttributeSet();
      Enumeration enumAttrs = findAttrs.getAttributes();

      /* Iterate through each attribute. */
      while ( enumAttrs.hasMoreElements() ) {
         LDAPAttribute anAttr = (LDAPAttribute)enumAttrs.nextElement();

         /* Get and print the attribute name. */
         String attrName = anAttr.getName();
         System.out.println( attrName );

         /* Get the values of the attribute. */
         Enumeration enumVals = anAttr.getStringValues();

         /* Get and print each value. */
         if ( enumVals == null ) {
            System.out.println( "\tNo values found." );
            continue;
         }
         while ( enumVals.hasMoreElements() ) {
            String aVal = ( String )enumVals.nextElement();
            System.out.println( "\t" + aVal );
         }
      }
   }
```

```
}
catch( LDAPException e ) {
   System.out.println( "Error: " + e.toString() );
}
...
```

Determining If the Server Supports LDAP v3

You can determine what version an LDAP server supports by getting the `supportedLDAPVersion` attribute from the root DSE. This attribute should contain the value 3. (It may also contain other values, such as 2, so you may want to check through the values of this attribute.)

Note that you do not need to authenticate or bind (see "Binding and Authenticating to an LDAP Server" for details) before searching the directory. Unlike the LDAP v2 protocol, the LDAP v3 protocol states that clients do not need to bind to the server before performing LDAP operations.

The following section of code connects to an LDAP server and determines whether or not that server supports the LDAP v3 protocol.

```
...
import netscape.ldap.*;
import java.util.*;
...
/* Create a new connection. */
LDAPConnection ld = new LDAPConnection();
String hostname = "localhost";
int portnumber = LDAPv2.DEFAULT_PORT;
boolean supportsV3 = false;

try {
   /* Connect to the LDAP server. */
   ld.connect( 3, hostname, portnumber );

   /* Get the root DSE by doing a search where:
         - The scope is SCOPE_BASE
         - The base is ""
         - The search filter is "(objectclass=*)"
   */
   int MY_SCOPE = LDAPv2.SCOPE_BASE;
   String MY_FILTER = "(objectclass=*)";
   String MY_SEARCHBASE = "";
   LDAPSearchResults res = ld.search( MY_SEARCHBASE,
      MY_SCOPE, MY_FILTER, null, false );
```

```
   /* There should be only one entry in the results (the root DSE). */
   while ( res.hasMoreElements() ) {
      LDAPEntry findEntry = (LDAPEntry)res.nextElement();

      /* Get the supportedLDAPVersion attribute. */
      LDAPAttribute versionAttr =
         findEntry.getAttribute( "supportedLDAPVersion" );

      /* Check if "3" is one of the supported LDAP versions. */
      Enumeration enumVals = versionAttr.getStringValues();
      if ( enumVals == null ) {
         System.out.println( "\tNo values found." );
         continue;
      }
      while ( enumVals.hasMoreElements() ) {
         String aVal = ( String )enumVals.nextElement();
         if ( aVal.equalsIgnoreCase( "3" ) ) {
            supportsV3 = true;
            break;
         }
      }
   }
}
catch( LDAPException e ) {
   System.out.println( "Error: " + e.toString() );
}
if ( supportsV3 ) {
   System.out.println( "This server supports LDAP v3." );
} else {
   System.out.println( "This server does not support LDAP v3." );
}
...
```

Getting Schema Information

In the LDAP v3 protocol, you can get and modify the schema for an LDAP server over the LDAP protocol. This section discusses the classes and methods that you can use to do this.

- “Overview: Schema Over LDAP”
- “Getting the Schema for an LDAP Server”
- “Working with Object Class Descriptions”
- “Working with Attribute Type Descriptions”

- “Working with Matching Rule Descriptions”
- “Example of Working with the Schema”

Overview: Schema Over LDAP

An entry can specify the schema that defines the object classes, attributes, and matching rules used by the directory. This entry is called the subschema entry.

To find the DN of the subschema entry, get the `subschemaSubentry` operational attribute from the root DSE or from any entry. (See “Specifying the Attributes to Retrieve” for details.) For example, in the root DSE for the Netscape Directory Server 4.1, the `subschemaSubentry` attribute specifies the location of the subschema entry.

The subschema entry itself can have the following attributes:

- `objectClasses` specifies the object class definitions in the schema. Each value of this attribute is an object class that is known to the server.
- `attributeTypes` specifies the attribute type definitions in the schema. Each value of this attribute is an attribute type that is known to the server.
- `matchingRules` specifies the matching rule definitions in the schema. Each value of this attribute is a matching rule that is known to the server.
- `matchingRuleUse` specifies the use of a matching rule in the schema. Each value of this attribute is a matching rule use description. A matching rule use description specifies the OIDs of the attributes that can be used with this extensible matching rule.

In the Netscape Directory SDK for Java, the schema and elements in the schema (object classes, attribute types, matching rules, and the use of matching rules) are represented by classes in the `netscape.ldap` package. The following table lists these classes.

Table 11.2 Classes that represent the schema and schema elements

Class Name	Description
`LDAPSchema`	The schema used by an LDAP server.
`LDAPSchemaElement`	Base class that represents a generic element in the schema.
`LDAPObjectClassSchema`	An object class description in the schema.
`LDAPAttributeSchema`	An attribute type description in the schema.
`LDAPMatchingRuleSchema`	A matching rule or matching rule use description in the schema.

Internally, these classes and their methods get and manipulate the subschema entry using standard LDAP operations, such as search and modify.

Getting the Schema for an LDAP Server

To get the schema for an LDAP v3 server, construct a new `LDAPSchema` object. Then, invoke the `fetchSchema` method, passing in an `LDAPConnection` object.

For example:

```
...
import netscape.ldap.*;
...
LDAPConnection ld = new LDAPConnection();

/* Construct a new LDAPSchema object to hold
   the schema that you want to retrieve. */
LDAPSchema dirSchema = new LDAPSchema();
try {
   ld.connect( hostname, portnumber, bindDN, bindPW );
   /* Get the schema from the Directory. Anonymous access okay. */
   dirSchema.fetchSchema( ld );
   ...
} catch ( Exception e ) {
   System.err.println( e.toString() );
}
...
```

Working with Object Class Descriptions

In the LDAP Java classes, the object class descriptions in a schema are represented by objects of the `LDAPObjectClassSchema` class.

To get the object class descriptions from the schema, you can invoke one of the following methods:

- To get an enumeration of `LDAPObjectClassSchema` objects representing the object classes in the schema, invoke the `getObjectClasses` method.
- To get a specific object class description, invoke the `getObjectClass` method and pass in the name of the object class.
- To get an enumeration of the names of object classes in the schema, invoke the `getObjectClassNames` method.

An object class description consists of the following information, which you can retrieve by invoking methods of the `LDAPObjectClassSchema` object:

- an OID identifying the object class (get this by invoking the `getOID` method, which is inherited from the `LDAPSchemaElement` base class)
- a name identifying the object class (get this by invoking the `getName` method, which is inherited from the `LDAPSchemaElement` base class)
- a description of the object class (get this by invoking the `getDescription` method, which is inherited from the `LDAPSchemaElement` base class)
- the name of the parent object class (get this by invoking the `getSuperior` method)
- the list of attribute types that are required in this object class (get this by invoking the `getRequiredAttributes` method)
- the list of attribute types that are allowed (optional) in this object class (get this by invoking the `getOptionalAttributes` method)

To add an object class description to the schema, construct a new `LDAPObjectClassSchema` object. You can specify the pieces of information in the object as individual arguments or in a description formatted according to RFC 2252, Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions (`http://www.ietf.org/rfc/rfc2252.txt`).

Then, you can either invoke the `add` method of this object (inherited from the `LDAPSchemaElement` base class) or the `addObjectClass` method of the `LDAPSchema` object.

To remove an object class description, you can invoke the `remove` method of this object (inherited from the `LDAPSchemaElement` base class).

Working with Attribute Type Descriptions

In the LDAP Java classes, the attribute type descriptions in a schema are represented by objects of the `LDAPAttributeSchema` class.

To get the attribute type descriptions from the schema, you can invoke one of the following methods:

- To get an enumeration of `LDAPAttributeSchema` objects representing the attribute types in the schema, invoke the `getAttributes` method.
- To get a specific attribute type description, invoke the `getAttribute` method and pass in the name of the attribute type.
- To get an enumeration of the names of attribute types in the schema, invoke the `getAttributeNames` method.

An attribute type description consists of the following information, which you can retrieve by invoking methods of the `LDAPAttributeSchema` object:

- an OID identifying the attribute type (get this by invoking the `getOID` method, which is inherited from the `LDAPSchemaElement` base class)
- a name identifying the attribute type (get this by invoking the `getName` method, which is inherited from the `LDAPSchemaElement` base class)
- a description of the attribute type (get this by invoking the `getDescription` method), which is inherited from the `LDAPSchemaElement` base class)
- the syntax used by the attribute type (get this by invoking the `getSyntax` method)

- an indicator of whether the attribute type is single-valued or multi-valued (get this by invoking the `isSingleValued` method)

To add an attribute type description to the schema, construct a new `LDAPAttributeSchema` object. You can specify the pieces of information in the object as individual arguments or in a description formatted according to RFC 2252, Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions (`http://www.ietf.org/rfc/rfc2252.txt`).

Then, you can either invoke the `add` method of this object (inherited from the `LDAPSchemaElement` base class) or the `addAttribute` method of the `LDAPSchema` object.

To remove an attribute type description, you can invoke the `remove` method of this object (inherited from the `LDAPSchemaElement` base class).

Working with Matching Rule Descriptions

In the LDAP Java classes, the matching rule descriptions and matching rule "use" descriptions in a schema are represented by objects of the `LDAPMatchingRuleSchema` class.

To get the matching rule descriptions from the schema, you can invoke one of the following methods:

- To get an enumeration of `LDAPMatchingRuleSchema` objects representing the matching rules in the schema, invoke the `getMatchingRules` method.
- To get a specific matching rule description, invoke the `getMatchingRule` method and pass in the name of the matching rule.
- To get an enumeration of the names of matching rules in the schema, invoke the `getMatchingRuleNames` method.

A matching rule description consists of the following information, which you can retrieve by invoking methods of the `LDAPMatchingRuleSchema` object:

- an OID identifying the matching rule (get this by invoking the `getOID` method, which is inherited from the `LDAPSchemaElement` base class)

- a name identifying the matching rule (get this by invoking the `getName` method, which is inherited from the `LDAPSchemaElement` base class)
- a description of the matching rule (get this by invoking the `getDescription` method, which is inherited from the `LDAPSchemaElement` base class)
- the syntax of the matching rule (get this by invoking the `getSyntax` method, which is inherited from the `LDAPAttributeSchema` class)

To add a matching rule description to the schema, construct a new `LDAPMatchingRuleSchema` object. You can specify the pieces of information in the object as individual arguments or in a description formatted according to RFC 2252, Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions (`http://www.ietf.org/rfc/rfc2252.txt`).

Then, you can either invoke the `add` method of this object (inherited from the `LDAPSchemaElement` base class) or the `addMatchingRule` method of the `LDAPSchema` object.

To remove a matching rule description, you can invoke the `remove` method of this object (inherited from the `LDAPSchemaElement` base class).

Example of Working with the Schema

The following section of code illustrates how to get the schema and how to add object classes and attribute types.

```
...
import netscape.ldap.*;
import java.util.*;
...
public class GetSchema {
   public static void main( String[] args ) {
      LDAPConnection ld = new LDAPConnection();
      String hostname = "localhost";
      int portnumber = LDAPv2.DEFAULT_PORT;
      String bindDN = "cn=Directory Manager";
      String bindPW = "23skidoo";

      /* Construct a new LDAPSchema object to hold
         the schema that you want to retrieve. */
      LDAPSchema dirSchema = new LDAPSchema();
      try {
         ld.connect( hostname, portnumber, bindDN, bindPW );
```

```
/* Get the schema from the Directory. Anonymous access okay. */
dirSchema.fetchSchema( ld );

/* Get and print the def. of the inetOrgPerson object class. */
LDAPObjectClassSchema objClass =
   dirSchema.getObjectClass( "inetOrgPerson" );
if ( objClass != null )
   System.out.println( "inetOrgPerson := " +
      objClass.toString() );

/* Get and print the def. of the userPassword attribute. */
LDAPAttributeSchema attrType =
   dirSchema.getAttribute( "userpassword" );
if ( attrType != null )
   System.out.println( "userPassword := " +
      attrType.toString() );

/* Add a new object class. */
String[] requiredAttrs = {"cn", "mail"};
String[] optionalAttrs = {"sn", "phoneNumber"};
LDAPObjectClassSchema newObjClass = new LDAPObjectClassSchema(
   "newInetOrgPerson", "1.2.3.4.5.6.7", "top", "Experiment",
   requiredAttrs, optionalAttrs );

/* Add the new object class to the schema. */
newObjClass.add( ld );

/* Create a new attribute type "hairColor". */
LDAPAttributeSchema newAttrType = new LDAPAttributeSchema(
   "hairColor", "1.2.3.4.5.4.3.2.1", "Blonde, red, etc",
   LDAPAttributeSchema.cis, false );

/* Add the new attribute type to the schema. */
newAttrType.add( ld );

/* Fetch schema again from the server to verify that the
   changes were made. */
dirSchema.fetchSchema( ld );

/* Get and print the new attribute type. */
newAttrType = dirSchema.getAttribute( "hairColor" );
if ( newAttrType != null )
   System.out.println( "hairColor := " +
      newAttrType.toString() );

/* Get and print the new object class. */
newObjClass = dirSchema.getObjectClass( "newInetOrgPerson" );
if ( newObjClass != null )
   System.out.println( "newInetOrgPerson := " +
```

```
                newObjClass.toString() );

            ld.disconnect();
         } catch ( Exception e ) {
            System.err.println( e.toString() );
            System.exit( 1 );
         }
         System.exit( 0 );
      }
   }
   ...
```

Chapter

12

Connecting Over SSL

This chapter describes the process of enabling an LDAP client to connect to an LDAP server over the Secure Sockets Layer (SSL) protocol. The chapter covers the procedures for connecting to an LDAP server and authenticating.

The chapter includes the following sections:

- "How SSL Works with the Netscape Directory SDK for Java"
- "Prerequisites for Connecting Over SSL"
- "Connecting to the Server Over SSL"
- "Using Certificate-Based Client Authentication"

How SSL Works with the Netscape Directory SDK for Java

The Netscape Directory SDK for Java includes classes and methods to enable your application to connect to an LDAP server over a Secure Sockets Layer (SSL).

Understanding SSL

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. For more information on SSL, see:

- The SSL 3.0 Protocol Specification (http://home.netscape.com/eng/ssl3/ssl-toc.html)

The Netscape Directory SDK for Java supports SSL 3.0. Note that SSL is not supported by all LDAP servers.

SSL Over LDAP

When an LDAP client connects to an LDAP server over SSL, the LDAP server identifies itself by sending its certificate to the LDAP client. The LDAP client needs to determine whether or not the certificate authority (CA) who issued the certificate is trusted.

The LDAP server may also request that the client send a certificate to authenticate itself. (This process is called certificate-based client authentication.)

After receiving the client's certificate, the LDAP server determines whether or not the CA who issued the certificate is trusted. If the CA is trusted, the server uses the subject name in the certificate to determine if the client has access rights to perform the requested operation.

In order to use SSL, you need a certificate database to hold the CA certificate and (if certificate-based client authentication is used) the client's certificate. For details, see "Prerequisites for Connecting Over SSL".

Interfaces and Classes for SSL

The Netscape Directory SDK for Java includes the `LDAPSocketFactory` interface, which describes a single method, `makeSocket`, that returns a socket to a given server (specified by a host name and port number). To establish an SSL connection, you need to create an object of a class that implements this interface.

Note that the classes that implement this interface in the Netscape Directory SDK for Java rely on a separate class that implements SSL sockets. In the constructors for object that implement `LDAPSocketFactory`, you typically need to specify the name of a class that implements SSL sockets.

The following classes implement this interface:

- `LDAPSSLSocketFactory`

  Use this class if you are using the `netscape.net.SSLSocket` class (which is provided with Netscape Communicator 4.05 and more recent versions) to implement SSL sockets. You can also use this class if the class that implements SSL sockets extends the `Socket` object.

- `LDAPSSLSocketWrapFactory`

  Use this class if the class that implements SSL sockets does not extend the `Socket` object. The `LDAPSSLSocketWrapFactory` class wraps your SSL socket implementation class in a class that does extend the `Socket` object.

You can construct an object of one of these factory classes and pass the factory object to the constructor for the `LDAPConnection` object to identify the socket factory that you want used for the connection.

Prerequisites for Connecting Over SSL

The LDAP Java classes that enable you to connect over SSL rely assume the following:

- Your client has access to a Netscape certificate database.

  If you are running your client as an applet in a Netscape Navigator browser, you can use this certificate database to determine if you trust the certificate sent from the server.

- The database that you are using contains any one of the following:

  - the certificate of the certificate authority (CA) that issued the server's certificate
  - if the certificate authorities (CAs) are organized in a hierarchy, the certificate of any of the CAs in the hierarchy
  - the certificate of the LDAP server

- The CA certificate is marked as "trusted" in the certificate database.
- If you plan to use certificate-based client authentication, you also need the following:
  - a client certificate (issued by a CA trusted by the LDAP server) in the certificate database
  - a public/private key pair in a Netscape key file (this can be either the `key.db` file used by Netscape Navigator or the `<alias>-key.db` file used by Netscape servers)

Essentially, when your client sends an initial request to the secure LDAP server, the server sends its certificate back to your client. Your client determines which CA issued the server's certificate and searches the certificate database for the certificate of that CA.

If your client cannot find the CA certificate or if the CA certificate is marked as "not trusted," your client refuses to connect to the server.

If you are using certificate-based client authentication, your client retrieves its certificate from the certificate database and sends it to the server for authentication. The server determines which CA issued the client's certificate and searches its certificate database for the certificate of that CA.

If the server cannot find the CA certificate or if the CA certificate is marked as "not trusted," the server refuses to authenticate your client.

Connecting to the Server Over SSL

To connect to an LDAP server using SSL, do the following:

1. Construct a new `LDAPSSLSocketFactory` object or a new `LDAPSSLSocketWrapFactory` object.

   This object represents the SSL socket factory that will be used to create the sockets for establishing connections with the LDAP server.

   The constructors for these classes allow you to specify the name of the class that will be used to create the actual sockets.

- For the `LDAPSSLSocketFactory` constructor, you should specify a class that implements the `javax.net.ssl.SSLSocket` interface. By default, if you do not specify a class, the `netscape.net.SSLSocket` class is used. This class is included with Netscape Communicator 4.05.
- If the SSL socket class does not extend the `Socket` class (for example, if it just extends the `Object` class), use the `LDAPSSLSocketWrapFactory` constructor.

2. Pass the object you constructed to the `LDAPConnection` constructor.

   When first establishing a connection to the LDAP server, the `makeSocket` method of the specified object will be used to construct the socket.

Using Certificate-Based Client Authentication

Some LDAP servers may be configured to use certificate-based client authentication. A server may request that your client send a certificate to identify itself.

Using the Netscape Directory SDK for Java, you can set up your client to perform certificate-based authentication in either of the following situations:

- Your client is an applet running in a Netscape browser.
- Your client is using a class that implements the `LDAPSocketFactory` interface and supports certificate-based client authentication.

Note the following:

- The `LDAPSSLSocketWrapFactory` class currently does not support certificate-based client authentication.
- The `LDAPSSLSocketFactory` class relies on the Netscape browser to support certificate-based client authentication. This class does not support the use of certificates for authentication outside the browser (for example, if your client is a stand-alone Java application).

To enable an applet to use certificate-based client authentication, do the following:

1. Construct a new `LDAPSSLSocketFactory` object.

2. Invoke the `enableClientAuth` method of the object to enable certificate-based client authentication.

3. Pass the object you constructed to the `LDAPConnection` constructor.

Chapter

13

Working with LDAP Controls

This chapter explains how LDAP controls work and how to use the LDAP controls that are supported by the Netscape Directory Server.

The chapter includes the following sections:

How LDAP Controls Work

The LDAP v3 protocol (documented in RFC 2251, "Lightweight Directory Access Protocol (v3)") allows clients and servers to use controls as a mechanism for extending an LDAP operation. A control is a way to specify additional information as part of a request and a response.

For example, a client can send a control to a server as part of a search request to indicate that the server should sort the search results before sending the results back to the client.

Servers can also send controls back to clients. For example, the Netscape Directory Server sends a control back to a client during the authentication process if the client's password has expired or is going to expire.

A control specifies the following information:

- A unique object identifier (OID)
- An indication of whether or not the control is critical to the operation
- Optional data related to the control (for example, for the server-side sorting control, the attributes used for sorting search results)

The OID identifies the control. If you plan to use a control, you need to make sure that the server supports the control. (See "Determining the Controls Supported By the Server" for details.)

When your client includes a control in a request for an LDAP operation, the server may respond in one of the following ways:

- If the server supports this control and if the control is appropriate to the operation, the server should make use of the control when performing the operation.
- If the server does not support the control type or if the control is not appropriate, the server should do one of the following:
    - If the control is marked as critical to the operation, the server should not perform the operation and should send an "unavailable critical extension" result code. When receiving this result code, your client throws an `LDAPException` with the result code `LDAPException.UNAVAILABLE_CRITICAL_EXTENSION`.

- If the control is marked as not critical to the operation, the server should ignore the control and should proceed to perform the operation.

Note that servers can also send controls back to clients.

There are two types of controls:

- Server controls can be included in requests sent by clients and in responses sent by servers.
- Client controls affect the behavior of the LDAP Java classes only and are never sent to the server. (At this point in time, no client controls are supported in the Netscape Directory SDK for Java.)

The next section describes how controls are implemented in the LDAP Java classes and which classes and methods you can use to create, send, and parse data from LDAP controls.

Using Controls in the LDAP Java Classes

In the LDAP Java classes, a control is represented by an object of the `LDAPControl` class.

To include a control in a request, you should do the following:

1. Invoke the `getSearchConstraints` method of the `LDAPConnection` object to get a clone of `LDAPSearchConstraints` for this connection.
2. Invoke the `setServerControls` method of the cloned constraints object, passing in the `LDAPControl` object that represents the control you want to include.
3. Invoke the appropriate method to perform the LDAP operation, passing in the constraints object. For example, if you are performing a search, invoke the `search` method and pass the search constraints as an argument.

(You can also include controls by invoking the `setServerControls` method for the default set of search constraints or by invoking the `setOption` method to set the `LDAPv3.SERVERCONTROLS` option. Note, however, that these controls

will be sent to the server with every request. In general, controls tend to be specific to a type of operation, so you should only include a control in a request for the operation that it applies to.)

You can then retrieve data from the returned controls through accessor methods in the `LDAPControl` object.

The rest of this chapter explains how to use the LDAP Java classes to send and retrieve specific types of controls.

Determining the Controls Supported By the Server

According to the LDAP v3 protocol, servers should list any controls that they support in the `supportedControl` attribute in the root DSE.

The following table lists some of the OIDs for server controls.

Table 13.1 LDAP v3 Server Controls

OID of Control and Defined Constant	Description of Control
2.16.840.1.113730.3.4.2 `netscape.ldap.LDAPControl.MANAGEDSAIT`	"Manage DSA IT" control (see “Using the Manage DSA IT Control” for details)
2.16.840.1.113730.3.4.3 `netscape.ldap.controls.LDAPPersistSearchControl.PERSISTENTSEARCH`	"Persistent search" control (see “Using the Persistent Search Control” for details)
2.16.840.1.113730.3.4.4 `netscape.ldap.LDAPControl.PWEXPIRED`	"Password expired" control (see “Using Password Policy Controls” for details)
2.16.840.1.113730.3.4.5 `netscape.ldap.LDAPControl.PWEXPIRING`	"Password expiration warning" control (see “Using Password Policy Controls” for details)
2.16.840.1.113730.3.4.9 `netscape.ldap.controls.LDAPVirtualListControl.VIRTUALLIST`	"Virtual list view" control (see “Using the Virtual List View Control” for details)

Table 13.1 LDAP v3 Server Controls

OID of Control and Defined Constant	Description of Control

2.16.840.1.113730.3.4.12
`netscape.ldap.controls.`
`LDAPProxiedAuthControl.`
`PROXIEDAUTHREQUEST`
	"Proxied authorization" control (see "Using the Proxied Authorization Control" for details)

1.2.840.113556.1.4.473
`netscape.ldap.controls.`
`LDAPSortControl.SORTREQUEST`
	"Server-side sorting" control (see “Using the Server-Side Sorting Control” for details)

The following example is a simple command-line program that searches for the root DSE and prints the values of the `supportedControl` attribute.

```
...
import netscape.ldap.*;
import netscape.ldap.controls.*;
import java.util.*;
...
public class ListCtrl {
   public static void main( String[] args ) {

      /* Hashtable mapping OIDs of known controls to descriptions of
         each control. */
      Hashtable knownControls = new Hashtable();
      knownControls.put( LDAPSortControl.SORTREQUEST,
         "Sort control" );
      knownControls.put( LDAPControl.MANAGEDSAIT,
         "ManageDsaIT control" );
      knownControls.put( LDAPPersistSearchControl.PERSISTENTSEARCH,
         "Persistent Search control" );
      knownControls.put( LDAPControl.PWEXPIRED,
         "Password Expiration Notification control" );
      knownControls.put( LDAPControl.PWEXPIRING,
         "Password Expiration Warning control" );
      knownControls.put( LDAPVirtualListControl.VIRTUALLIST,
         "Virtual List View control" );
      knownControls.put( LDAPProxiedAuthControl.PROXIEDAUTHREQUEST,
         "Proxied Authorization control" );

      LDAPConnection ld = new LDAPConnection();
      String hostname = "localhost";
      int portnumber = LDAPv2.DEFAULT_PORT;
      try {

         /* Connect and authenticate to server */
         ld.connect( 3, hostname, portnumber );
```

```
/* The list of supported controls can be retrieved by
   getting the root DSE.  To get the root DSE, you need
   to do a search where:
   - The scope is SCOPE_BASE
   - The base is ""
   - The search filter is "(objectclass=*)"
   The values of the supportedControl attribute of the
   resulting entry are the OIDs of the supported controls. */
int MY_SCOPE = LDAPv2.SCOPE_BASE;
String MY_FILTER = "(objectclass=*)";
String MY_SEARCHBASE = "";
String getAttrs[] = { "supportedControl" };
LDAPSearchResults res = ld.search( MY_SEARCHBASE,
   MY_SCOPE, MY_FILTER, getAttrs, false );

/* There should only be one entry found. */

/* Get the attributes of this entry. */
LDAPEntry DSE = (LDAPEntry)res.nextElement();
LDAPAttributeSet findAttrs = DSE.getAttributeSet();
Enumeration enumAttrs = findAttrs.getAttributes();

/* Print each attribute returned
   (this should only be the supportedControl attribute) */
while ( enumAttrs.hasMoreElements() ) {
   LDAPAttribute anAttr =
      (LDAPAttribute)enumAttrs.nextElement();
   String attrName = anAttr.getName();
   System.out.println( attrName );

   /* Get the values of this attribute. */
   Enumeration enumVals = anAttr.getStringValues();
   if ( enumVals == null ) {
      System.out.println( "\tNo values." );
      continue;
   }
   while ( enumVals.hasMoreElements() ) {
      String aVal = ( String )enumVals.nextElement();

      /* Each value should be the OID of a control. Look up the
         description corresponding to the OID. */
      String aDesc = ( String )knownControls.get( aVal );
      if ( aDesc != null ) {
         System.out.println( "\t" + aDesc+ " (" + aVal + ")" );
      } else {
         System.out.println( "\t" + aVal );
      }
   }
}
```

```
        }
        catch( LDAPException e ) {
              System.out.println( "Error: " + e.toString() );
        }
        try {
           ld.disconnect();
        }
        catch( LDAPException e ) {
            System.exit(1);
        }
        System.exit(0);
    }
}
...
```

Using the Server-Side Sorting Control

The control with the OID 1.2.840.113556.1.4.473 (or the constant `netscape.ldap.LDAPControl.SORTREQUEST`) is a server-side sorting control. When you send a search request with this control to the server, the server should sort the results before sending them back to you.

The server-side sorting control is described in the Internet-Draft "LDAP Control Extension for Server Side Sorting of Search Results." This document is available at: `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-sorting-02.txt`

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

This section covers the following topics:

- “Specifying the Sort Order”
- “Creating the Control”
- “Performing the Search”
- “Interpreting the Results”
- “Known Problems with Server Sorting”
- “Example of Using the Server-Sorting Control”

Specifying the Sort Order

To specify the sort order of the results, construct one or more `LDAPSortKey` objects. Each object represents a sort key generated from a string in the following format:

```
[-]<attrname>[:<matchingruleoid>]
```

`<attrname>` is the name of the attribute that you want to sort by. `<matchingruleoid>` is the optional OID of a matching rule that you want to use for sorting. The minus sign indicates that the results should be sorted in reverse order for that attribute.

For example, the following string specifies that results should be sorted by first name ("givenname") in descending order:

```
"-givenname"
```

Pass this string to the `LDAPSortKey` constructor to create a sort key. For example:

```
LDAPSortKey sortByFirstNameReverse = new LDAPSortKey( "-givenname" );
```

To sort by more than one attribute, construct more than one `LDAPSortKey` object and create an array of the objects.

For example, suppose you want to sort the result by last name ("sn") in ascending order. If two or more entries have the same last name, you want to sort the result by first name ("givenname") in ascending order.

To specify this sort order, you construct two `LDAPSortKey` objects and create an array:

```
...
LDAPSortKey sortByLastName = new LDAPSortKey( "sn" );
LDAPSortKey sortByFirstName = new LDAPSortKey( "givenname" );
LDAPSortKey[] sortOrder = { sortByLastName, sortByFirstName };
...
```

If you are using the Netscape Directory Server 3.*x*, keep the following in mind:

- For attributes that have multiple values, the Netscape Directory Server 3.*x* will use the "highest" value when sorting.

  For example, suppose you choose to sort by the ou attribute. If an entry has "Accounting" and "People" as values of the ou attribute, the server will use "People" when sorting this entry.

  On the other hand, if the values are "People" and "Product Development", the server will use "Product Development" when sorting the entry.

- If an entry does not contain the attribute used for sorting, the Netscape Directory Server 3.*x* sorts that entry before other entries that do contain the attribute.

  For example, suppose the entry for Barbara Jensen does not contain the ou attribute and the entry for Kurt Jensen does contain the ou attribute. If the server is sorting by the ou attribute, Barbara Jensen's entry will come before Kurt Jensen's entry in the sorted results.

Creating the Control

Next, to create the server-side sorting control, construct a new `LDAPSortControl` object. Pass the `LDAPSortKey` object (or the array of `LDAPSortKey` objects) to the `LDAPSortControl` constructor.

In the constructor, you can also specify whether or not the control is critical to the search operation. If the control is marked as critical and the server cannot sort the results, the server should not send back any entries. (See "Interpreting the Results" for more information on the effect of a critical control.)

For example, the following section of code creates a server-side sorting control and specifies that the control is critical to the search operation:

```
LDAPSortKey sortOrder = new LDAPSortKey( "-givenname" );
LDAPSortControl sortCtrl = new LDAPSortControl( sortOrder, true );
```

Performing the Search

To specify that you want the server to sort the results, do the following:

1. Get a clone of `LDAPSearchConstraints` for the current connection by invoking the `getSearchConstraints` method of the `LDAPConnection` object.

2. Invoke the `setServerControls` method for the copied `LDAPSearchConstraints` object, and pass in the `LDAPSortControl` object that you have constructed.

3. Invoke the `search` method of the `LDAPConnection` object, passing in the `LDAPSearchConstraints` object.

   The server returns a result for the search operation and a response control. The response control indicates the success or failure of the sorting.

4. Invoke the `getResponseControls` method of the `LDAPSearchResults` object to retrieve any controls sent back by the server in response to the search.

   Response controls are passed back as an array of `LDAPControl` objects.

5. Examine the type of each returned control. If a control is an instance of `LDAPSortControl`, you can read the result code for the sorting operation using the `getResultCode` method.

   If the sorting operation failed, the server may also return the name of the attribute that caused the failure. You can read the name of this attribute with the `getFailedAttribute` method.

The server can return the following result codes that apply to the sorting operation.

Table 13.2 LDAP result codes for sorting search results

Result Code	Description
`LDAPException.SUCCESS`	The results were sorted successfully.
`LDAPException.OPERATION_ERROR`	An internal server error occurred.
`LDAPException.TIME_LIMIT_EXCEEDED`	The maximum time allowed for a search was exceeded before the server finished sorting the results.

Table 13.2 LDAP result codes for sorting search results

Result Code	Description
LDAPException. STRONG_AUTH_REQUIRED	The server refused to send back the sorted search results because it requires you to use a stronger authentication method.
LDAPException. ADMIN_LIMIT_EXCEEDED	There are too many entries for the server to sort.
LDAPException. NO_SUCH_ATTRIBUTE	The sort key list specifies an attribute that does not exist.
LDAPException. INAPPROPRIATE_MATCHING	The sort key list specifies a matching rule that is not recognized or appropriate
LDAPException. INSUFFICIENT_ACCESS_RIGHTS	The server did not send the sorted results because the client has insufficient access rights
LDAPException.BUSY	The server is too busy to sort the results.
LDAPException. UNWILLING_TO_PERFORM	The server is unable to sort the results.
LDAPException.OTHER	This general result code indicates that the server failed to sort the results for a reason other than the ones listed above.

Interpreting the Results

The following table lists the kinds of results to expect from the LDAP server under different situations.

Does the Server Support the Sort Control?	Is the Sort Control Marked as Critical?	Are There any Other Conditions?	Results from the LDAP Server
No	No	N/A	The server ignores the sorting control and returns the entries unsorted.
No	Yes	N/A	The server does not send back any entries

Does the Server Support the Sort Control?	Is the Sort Control Marked as Critical?	Are There any Other Conditions?	Results from the LDAP Server
Yes	No	The server cannot sort the results using the specified sort key list.	• The server returns the entries unsorted. • The server sends back the sorting response control, which specifies the result code of the sort attempt and (optionally) the attribute type that caused the error.
Yes	Yes	The server cannot sort the results using the specified sort key list.	• The server does not send back any entries. • The server sends back the sorting response control, which specifies the result code of the sort attempt and (optionally) the attribute type that caused the error.
Yes	Yes or No (Does not affect the results)	The search itself failed.	• The server sends back a result code for the search operation. • The server does not send back the sorting response control.
Yes	Yes or No (Does not affect the results)	The server successfully sorted the entries.	• The server sends back the sorted entries. • The server sends back the sorting response control, which specifies the result code of the sort attempt (`LDAPException.SUCCESS`).

Known Problems with Server Sorting

The following problems may occur when using the server-side sorting control with the Netscape Directory Server 3.*x* (most of these problems are fixed in version 4.0 of the server):

- The server does not sort entries if the search filter consists of unindexed attributes.

  When processing a search request, the server generates a list of potential candidates from the indexes, sorts the candidate list, then compares each candidate against the search filter.

  If no indexes are applicable to the search, the candidate list consists of all entries. If the candidate lists consists of all entries, the server does not sort the list and instead sends back an `LDAP_UNWILLING_TO_PERFORM` result code.

  For example, suppose your search filter is "l=Sunnyvale". Since the "l" attribute (the "location" attribute) is not indexed by default, the server cannot use any indexes to narrow down the list of potential candidates, and all entries are considered candidates. If you attempt to sort by any attribute, the server will refuse to sort the entries.

- The server does not sort entries if you bind as the root DN and specify no time limit.

  Normally, if you bind as the root DN and specify no time limit, the server allows you an infinite time limit. As the server sorts the list of candidate entries during a search, the server checks to determine if the time limit has been exceeded. However, the server incorrectly calculates the amount of time allowed, so larger sets of candidates will not be sorted.

Example of Using the Server-Sorting Control

The following program uses the server-sorting control to get a list of all users in the directory, sorted in ascending order by last name, then in descending order by first name.

```
...
import netscape.ldap.*;
import netscape.ldap.controls.*;
import java.util.*;
...
public class SrchSort {
   public static void main( String[] args ) {
      LDAPConnection ld = new LDAPConnection();
      String hostname = "localhost";
      int portnumber = LDAPv2.DEFAULT_PORT;
```

```
int status = -1;

try {
   /* Connect to server */
   ld.connect( 3, hostname, portnumber, "", "" );

   /* Search for last names that start with "Wa". */
   String MY_FILTER = "sn=Wa*";
   String MY_BASE = "o=Airius.com";
   String[] attrs = { "sn", "givenname" };

   /* Create sort keys that specify the sort order. */
   LDAPSortKey sortByLastName = new LDAPSortKey( "sn" );
   LDAPSortKey sortByFirstName = new LDAPSortKey( "-givenname" );
   LDAPSortKey[] sortOrder = { sortByLastName, sortByFirstName };

   /* Create a server control using that sort key. */
   LDAPSortControl sortCtrl = new LDAPSortControl(sortOrder,
      true);

   /* Create search constraints to use that control. */
   LDAPSearchConstraints cons = ld.getSearchConstraints();
   cons.setServerControls( sortCtrl );

   /* Perform the search. */
   LDAPSearchResults res = ld.search( MY_BASE,
      LDAPv3.SCOPE_SUB, MY_FILTER, attrs, false, cons );

   /* Loop through the results until finished. */
   System.out.println( "Sorted Results from Server" );
   System.out.println( "==========================" );
   while ( res.hasMoreElements() ) {

      /* Get the next directory entry. */
      LDAPEntry findEntry = null;
      try {
         findEntry = res.next();

      /* Skip any referrals found for now. */
      } catch ( LDAPReferralException e ) {
         continue;
      } catch ( LDAPException e ) {
         System.out.println( "Error: " + e.toString() );
         continue;
      }

      /* Get the set of attributes for that entry. */
      LDAPAttributeSet findAttrs = findEntry.getAttributeSet();
      Enumeration enumAttrs = findAttrs.getAttributes();
```

```
   /* Iterate through the attributes. */
   while ( enumAttrs.hasMoreElements() ) {
      LDAPAttribute anAttr =
         (LDAPAttribute)enumAttrs.nextElement();

      /* Get the set of values for each attribute. */
      Enumeration enumVals = anAttr.getStringValues();
      if ( enumVals == null ) {
         System.out.println( "\tNo values." );
         continue;
      }

      /* Iterate through the values and print each value. */
      String aVal = (String)enumVals.nextElement();
      System.out.print( aVal );
      while (enumVals.hasMoreElements() ) {
         aVal = (String)enumVals.nextElement();
         System.out.print( ", " + aVal );
      }
      System.out.print( "\t\t" );
   }
System.out.println( "" );
}

/* Determine if the server sent a control back to you. */
LDAPControl[] returnedControls = res.getResponseControls();
if ( returnedControls != null ) {
   for ( int i=0; i<returnedControls.length; i++ ){
      if (!(returnedControls[i] instanceof LDAPSortControl)){
         continue;
      }
      LDAPSortControl sortRsp = (LDAPSortControl)
                                 returnedControls[i];
      int resultCode = sortRsp.getResultCode;

      /*Check if the result code indicated an error occurred.*/
      if ( resultCode != 0 ) {
         System.out.println( "Result code: " + resultCode );
         System.out.println(
            LDAPException.errorCodeToString( resultCode ) );

         /* If the server specified the attribute that
         caused the failure, print it out. */
         String failedAttr = sortRsp.getFailedAttribute();
         if ( failedAttr != null ) {
            System.out.println( "Failed on "+failedAttr );
         } else {
            System.out.println( "Server did not indicate which
               " + "attribute caused sorting to fail." );
         }
```

```
                }
             }
          }
          status = 0;
       }
       catch( LDAPException e ) {
          System.out.println( "Error: " + e.toString() );
       }

       /* Done, so disconnect */
       if ( (ld != null) && ld.isConnected() ) {
          try {
             ld.disconnect();
             System.out.println( "" );
          } catch ( LDAPException e ) {
             System.out.println( "Error: " + e.toString() );
          }
    }
       System.exit(status);
    }
}
...
```

Using the Persistent Search Control

The control with the OID 2.16.840.1.113730.3.4.3 (the constant `netscape.ldap.controls.LDAPPersistSearchControl.PERSISTENTSEARCH`) is the persistent search control. A persistent search (an ongoing search operation), which allows your LDAP client to get notification of changes to the directory.

The persistent search control is described in the Internet-Draft "Persistent Search: A Simple LDAP Change Notification Mechanism." This document is available at: `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-psearch-00.txt`

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

To use persistent searching for change notification, you create a "persistent search" control that specifies the types of changes that you want to track. You include the control in a search request. If an entry in the directory is changed,

the server determines if the entry matches the search criteria in your request and if the change is the type of change that you are tracking. If both of these are true, the server sends the entry to your client.

You can use this control in conjunction with an "entry change notification" control (see “Using the Entry Change Notification Control”).

The rest of this section describes how to create and use this control.

- “Creating the Control”
- “Performing the Search”
- “Example of Using the Persistent Search Control”

Creating the Control

To create a persistent search control, you construct a new `LDAPPersistSearchControl` object. When invoking the `LDAPPersistSearchControl` constructor, you can specify the following information:

- The type of change you want to track. You can specify any of the following (or any combination of the following using a bitwise OR operator):
  - `ADD` indicates that you want to track added entries
  - `DELETE` indicates that you want to track deleted entries
  - `MODIFY` indicates that you want to track modified entries
  - `MODDN` indicates that you want to track renamed entries
- A preference indicating whether you want the server to return all entries that initially matched the search criteria
- A preference indicating whether or not you want entry change notification controls included with every modified entry returned by the server

For example, the following section of code

```
...
/* Track all types of changes. */
int op = LDAPPersistSearchControl.ADD |
```

```
   LDAPPersistSearchControl.MODIFY |
   LDAPPersistSearchControl.DELETE |
   LDAPPersistSearchControl.MODDN;

/* Return only entries that have changed. */
boolean changesOnly = true;

/* Return an "entry change notification" control. */
boolean returnControls = true;

/* Mark the control as critical. */
boolean isCritical = true;

/* Create the control. */
LDAPPersistSearchControl persistCtrl = new
   LDAPPersistSearchControl( op, changesOnly,
   returnControls, isCritical );
...
```

Performing the Search

To specify that you want to start a persistent search, do the following:

1. Get a clone of `LDAPSearchConstraints` for the current connection by invoking the `getSearchConstraints` method of the `LDAPConnection` object.
2. Invoke the `setServerControls` method for the cloned `LDAPSearchConstraints` object, and pass in the `LDAPPersistSearchControl` object that you have constructed.
3. Invoke the `search` method of the `LDAPConnection` object, passing in the `LDAPSearchConstraints` object.

   The server returns matching entries as they change. If you specified that you wanted an "entry change notification" control included with each entry, you can get these controls from the server's results. For details, see “Using the Entry Change Notification Control”.

To end the persistent search, you can either invoke the `abandon` method of the `LDAPConnection` object to abandon the search operation, or you can invoke the `disconnect` method to disconnect from the server.

Example of Using the Persistent Search Control

The following program performs a persistent search and receives "entry change notification" controls from the server.

```
import netscape.ldap.*;
import netscape.ldap.controls.*;
import java.util.*;
public class SrchPrst implements Runnable {
   private String hostname;
   private int portnumber;
   public SrchPrst() {
   }

   public static void main( String[] args ) {
      /* Start up a new thread. */
      Thread th = new Thread( new SrchPrst(), "mainConn" );
      th.start();
      System.out.println( "Main thread started." );
   }

   public void run() {
      LDAPConnection ld = new LDAPConnection();
      String hostname = "localhost";
      int portnumber = LDAPv2.DEFAULT_PORT;
      try {
         /* Connect to server */
         ld.connect( 3, hostname, portnumber, "", "" );
         /* Specify that you want to track changes to all entries. */
         String MY_FILTER = "(objectclass=*)";
         String MY_BASE = "o=Airius.com";

         /* Create a persistent search control. */
         int op = LDAPPersistSearchControl.ADD |
            LDAPPersistSearchControl.MODIFY |
            LDAPPersistSearchControl.DELETE |
            LDAPPersistSearchControl.MODDN;
         boolean changesOnly = true;
         boolean returnControls = true;
         boolean isCritical = true;
         LDAPPersistSearchControl persistCtrl = new
               LDAPPersistSearchControl( op, changesOnly,
               returnControls, isCritical );

         /* Create search constraints to use that control. */
         LDAPSearchConstraints cons = ld.getSearchConstraints();
         cons.setServerControls( persistCtrl );
```

```
/* Start the persistent search. */
LDAPSearchResults res = ld.search( MY_BASE,
   LDAPv3.SCOPE_SUB, MY_FILTER, null, false, cons );

/* Loop through the results until finished. */
while ( res.hasMoreElements() ) {

   /* Print any entries that have changed. */
   System.out.println( "\n======= Changed Entry =======" );

   /* Get the next directory entry. */
   LDAPEntry findEntry = res.next();

   /* Get the set of attributes for that entry. */
   LDAPAttributeSet findAttrs = findEntry.getAttributeSet();
   Enumeration enumAttrs = findAttrs.getAttributes();

   /* Iterate through the attributes. */
   while ( enumAttrs.hasMoreElements() ) {
      LDAPAttribute anAttr =
         (LDAPAttribute)enumAttrs.nextElement();
      String attrName = anAttr.getName();
      System.out.println( "\t" + attrName );

      /* Get the set of values for each attribute. */
      Enumeration enumVals = anAttr.getStringValues();

      /* Iterate through the values and print each value. */
      while (enumVals.hasMoreElements() ) {
         String aVal = (String)enumVals.nextElement();
         System.out.println( "\t\t" + aVal );
      }
   }

   /* Get any entry change controls. */
   LDAPControl[] responseCtrls = res.getResponseControls();
   if ( responseCtrls != null ) {
      for ( int i=0; i<responseCtrls.length; i++ ){
         if (!(responseCtrls[i] instanceof
               LDAPEntryChangeControl)){
            continue;
         }
         LDAPEntryChangeControl entryCtrl =
            (LDAPEntryChangeControl) responseCtrls[i];

         /* Get information on the type of change made. */
         int changeType = entryCtrl.getChangeType();
         if ( changeType != -1 ) {
            System.out.print( "Change made: " );
            switch ( changeType ) {
```

```
                        case LDAPPersistSearchControl.ADD:
                           System.out.println( "Added new entry." );
                           break;
                        case LDAPPersistSearchControl.MODIFY:
                           System.out.println( "Modified entry." );
                           break;
                        case LDAPPersistSearchControl.DELETE:
                           System.out.println( "Deleted entry." );
                           break;
                        case LDAPPersistSearchControl.MODDN:
                           System.out.println( "Renamed entry." );
                           break;
                        }
                     }

                     /* Get the change log number, if present. */
                     int changeNumber = entryCtrl.getChangeNumber();
                     if ( changeNumber != -1 ) {
                        System.out.println( "Change log number: " +
                           changeNumber );
                     }

                     /* Get the previous DN of the entry,
                        if a modify DN operation was performed. */
                     String oldDN = entryCtrl.getPreviousDN();
                     if ( oldDN != null ) {
                        System.out.println( "Previous DN: " + oldDN );
                     }
                  } else {
                     System.out.println( "No entry change control." );
                  }
                  System.out.println( "\n" );
               }
            }
         }
         catch( LDAPException e ) {
            System.out.println( "Error: " + e.toString() );
         }
      }
   }
...
```

Using the Entry Change Notification Control

The control with the OID 2.16.840.1.113730.3.4.7 (or the constant `netscape.ldap.controls.LDAPEntryChangeControl.ENTRYCHANGED`) is the "entry change notification" control. These types of controls can be included with entries sent back from the server during a persistent search. (For more information on persistent searches, see “Using the Persistent Search Control”.)

The "entry change notification" control is described in the Internet-Draft "Persistent Search: A Simple LDAP Change Notification Mechanism." This document is available at: `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-psearch-00.txt`

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

The rest of this section describes how to use these types of controls.

- “Getting the Control”
- “Working with Change Log Numbers”

Getting the Control

To get an "entry change notification" control that is included with an entry, do the following:

1. As you retrieve each entry, invoke the `getResponseControls` method of the `LDAPConnection` object to retrieve any response controls sent back from the server.

   Response controls are passed back as an array of `LDAPControl` objects.

2. Pass this array of `LDAPControl` objects as an argument to the `LDAPPersistSearchControl.parseResponse` static method to retrieve the "entry change notification" control.

An "entry change notification" control is represented by an object of the `LDAPEntryChangeControl` class. To get data from this control, you can invoke the accessor methods, such as `getChangeNumber`, `getChangeType`, and `getPreviousDN`.

Working with Change Log Numbers

If the Netscape Directory Server is set up to be a supplier (for replicating changes in the directory to other servers), the server keeps a record of the changes made to the directory in a change log. Each record of a change has a number that identifies it in the log.

You can get the change number for a modified entry from the `LDAPEntryChangeControl` object.

If you want to look up the record for a particular change log number (to get more information about the change that took place), you can search for the record. In the Netscape Directory Server, the change log is represented by an entry in the directory, and individual change records are represented by entries in a subtree beneath the change log entry.

To determine the DN for the change log entry, search the root DSE and retrieve the `changelog` attribute. For example, the value of this attribute might be "cn=changelog", which is the DN for the change log entry.

Each change log record is an entry under the change log entry. The change log number is the value of the `changenumber` attribute of the record. To get a specific change log record, you can search the directory with the base DN "cn=changelog" and with the search filter "changenumber=<value>", where <value> is the change number of the record.

Note that you may need to authenticate as the directory manager to have access to these entries.

Using the Virtual List View Control

The control with the OID 2.16.840.1.113730.3.4.9 (or the constant `netscape.ldap.controls.LDAPVirtualListControl.VIRTUALLIST`) is a virtual list view control. When you send a search request with this control and with a server-side sorting control to the server, the server should sort the results and return the specified subset of entries back to your client.

The virtual list view control is described in the Internet-Draft "LDAP Extensions for Scrolling View Browsing of Search Results." This document is available at: `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldapv3-vlv-01.txt`

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

Although Netscape Directory Server 4.*x* supports this control, not all servers do. For information on determining if a server supports this or other LDAP v3 controls, see “Determining the Controls Supported By the Server”.

Note that after you set the list size by invoking the `setListSize` method of the `LDAPVirtualListControl` object, you need to invoke the `setRange` method to recreate the control using the new data. (`setRange` generates the BER-encoded request to be sent to the server; `setListSize` does not do this.)

Using the Manage DSA IT Control

The control with the OID 2.16.840.1.113730.3.4.2 (or the constant `netscape.ldap.LDAPControl.MANAGEDSAIT`) is the manage DSA IT control. You can use this control to manage search references in the directory.

The manage DSA IT control is described in the Internet-Draft "LDAP Control Extension for Server Side Sorting of Search Results." This document is available at: `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-sorting-00.txt`

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

To create this control, construct a new `LDAPControl` object. In the `LDAPControl` constructor, set the OID of the control to 2.16.840.1.113730.3.4.2.

When you add this control to the array of `LDAPControl` objects that you pass to a method that performs an LDAP operation, the server treats search references as ordinary entries.

Rather than returning a reference to you, the server returns the entry containing the reference. This allows your client application to manage search references in the directory.

Using Password Policy Controls

The Netscape Directory Server 3.0 and later versions use two server response controls to send information back to a client after an LDAP bind operation:

- The control with the OID 2.16.840.1.113730.3.4.4 (or the constant `netscape.ldap.LDAPControl.PWEXPIRED`) is the expired password control.

  This control is used if the server is configured to require users to change their passwords when first logging in and whenever the passwords are reset.

  If the user is logging in for the first time or if the user's password has been reset, the server sends this control to indicate that the client needs to change the password immediately.

  At this point, the only operation that the client can perform is to change the user's password. If the client requests any other LDAP operation, the server sends back an `LDAP_UNWILLING_TO_PERFORM` result code with an expired password control.

- The control with the OID 2.16.840.1.113730.3.4.5 (or the constant `netscape.ldap.LDAPControl.PWEXPIRING`) is the password expiration warning control.

  This control is used if the server is configured to expire user passwords after a certain amount of time.

  The server sends this control back to the client if the client binds using a password that will soon expire. If you invoke the `getValue` method for this `LDAPControl` object, the method returns the number of seconds before the password will expire.

To get these server response controls when binding, invoke the `getResponseControls` method of the `LDAPConnection` object after you attempt to authenticate to the server.

Using the Proxied Authorization Control

The control with the OID 2.16.840.1.113730.3.4.12 (or the constant `netscape.ldap.LDAPProxiedAuthControl.PROXIEDAUTHREQUEST`) allows LDAP clients to use different credentials, without rebinding, when executing LDAP operations. This is called proxied authorization.

For example, suppose we have a messaging server that stores its user profiles on an LDAP server. For certain types of requests the messaging server needs to use a DN and password other than its own. Doing this without proxied authorization requires the messaging server to rebind, using the different credentials, before executing each operation.

If the messaging server uses the proxied authorization control, it can act as the user when executing an operation, while only maintaining its own binding to the LDAP server. This drastically improves performance, especially when processing a large number of requests.

The control is described more fully in the Internet-Draft "LDAP Proxied Authorization Control." This document is available at: `http://www.ietf.org/internet-drafts/draft-weltman-ldapv3-proxy-03.txt`.

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

Chapter

14

Using SASL Authentication

This chapter describes the process of using a SASL mechanism to authenticate an LDAP client to an LDAP server.

The chapter includes the following sections:

- “Understanding SASL”
- “Preparing to Use SASL Authentication”
- “Using SASL in the Client”
- “For More Information”

Understanding SASL

The Simple Authentication and Security Layer (SASL) is an authentication method. It allows you to use mechanisms other than simple passwords and SSL for authenticating over connection-based protocols, such as LDAP.

All SASL mechanisms are registered with the Internet Assigned Numbers Authority (IANA). Included among these mechanisms are KERBEROS_V4, GSSAPI, and several others. The client implements these mechanisms through the use of mechanism drivers. These drivers are classes that contain the code required for authenticating over a given mechanism.

When a client attempts to authenticate to a Directory Server using the `LDAPConnection.authenticate` method, it can specify a list of SASL mechanisms to use. If the client does not specify any mechanisms, the SDK will query the server to find out which mechanisms it supports. If the SDK and the server have a common mechanism, authentication can occur.

If the server supports a requested mechanism, it responds with one or more challenges. In order to authenticate, the client must correctly respond to these challenges. This is handled transparently by the SDK using a mechanism driver.

If the server does not support any of the requested mechanisms, the SDK throws an `AuthenticationNotSupportedException`.

If the mechanism driver requires additional authentication data from the client it sends a `Callback` object to the client. To prepare for this, the client implements a `CallbackHandler` and passes it to the SDK. If the SASL mechanism needs to obtain additional credentials or notify the client of errors during the SASL negotiations, it calls the `CallbackHandler` object with `Callback` objects for each item to be processed. The `CallbackHandler` then decides how to proceed.

The Netscape Directory SDK for Java includes a package called `com.netscape.sasl` which contains the code necessary to perform all of the steps involved in SASL authentication.

Preparing to Use SASL Authentication

Before performing SASL authentication

- the LDAP server must support at least one SASL mechanism
- your client environment must support at least one of the SASL mechanisms supported by the server.

The rest of this section describes how to do this using the Netscape Directory SDK for Java and the Netscape Directory Server.

Supporting SASL on the Server

If you are running Netscape Directory Server 3.0 or later, you can write your own server plug-in to handle SASL authentication.

This pre-operation bind plug-in uses a registered SASL mechanism to

- get information from a SASL bind request
- create and send a SASL bind response back to the client.

  This response can take the form of a challenge requiring an answer from the client, an error message, or a success message indicating that authentication is complete.

For more information on how to write this plug-in, see "Defining Functions for Authentication" in the *Netscape Directory Server Plug-In Programmer's Guide*.

For more information on SASL mechanisms, see “For More Information” at the end of this chapter.

Supporting SASL on the Client

In order to authenticate over SASL, you will need to have a mechanism available in your SASL client package. If you have obtained a `ClientFactory` class that can produce a SASL mechanism which your server supports, you can name its package in your code.

There are two ways to do this. You can either

- request a SASL client and specify the package in the `javax.security.sasl.client.pkgs` property of its Hashtable

or

- set the package as the default factory for the session with `Sasl.setSaslClientFactory`.

For example, if you have a class called `mysecurity.sasl.ClientFactory` which is capable of producing a `SaslClient` object for one or more mechanisms, you could either write:

```
Hashtable props = new Hashtable();
props.put ( "javax.security.sasl.client.pkgs", "mysecurity.sasl" );
ld.authenticate( dn, props, cbh );
```

or

```
Sasl.setSaslClientFactory (new mysecurity.sasl.ClientFactory() );
ld.authenticate( dn, props, cbh );
```

The arguments are as follows:

Table 14.1 Arguments for LDAPConnection.authenticate (above)

Argument Name	Description
`dn`	Authentication DN
`props`	Any optional properties that the mechanism accepts. See Table 14.3 for details.
`cbh`	An instance of the CallbackHandler implemented in your application.

Implementing javax.security.auth.callback

Some SASL mechanisms require additional credentials during the authentication process. In order to provide this additional information, your SASL client may need to implement `Callback` objects and a `CallbackHandler` to list them. `Callback` and `CallbackHandler` are part of the `javax.security.auth.callback` package. The package is part of the Java Authentication and Authorization Service (JAAS) and is contained in the `jaas.jar` file.

To install the `javax.security.auth.callback` classes:

1. Locate the `jaas.jar` file

   The file is included in the `directory/java-sdk/ldapjdk/lib` directory of the Directory SDK for Java. You can also download the release version of these classes and all subsequent updates at

   `http://java.sun.com:8081/security/jaas/index.html`.

2. Add the `jaas.jar` file to your `CLASSPATH`

3. Import `javax.security.auth.callback.*` in your code.

The following is an example of `Callback` and `CallbackHandler` implementations.

```
class SampleCallbackHandler implements CallbackHandler {
   SampleCallbackHandler( String userName ) {
      userName = userName;
   }
   /** Invoke the requested Callback */
   public void invokeCallback(Callback[] callbacks)
      throws java.io.IOException, UnsupportedCallbackException {
         for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof TextOutputCallback) {
               // display the message according to the
               // specified STYLE
               TextOutputCallback toc =
                  (TextOutputCallback)callbacks[i];
               switch (toc.getStyle()) {
                  case TextOutputCallback.ERROR:
                  System.out.println("ERROR: " +
                     toc.getMessage());
                  break;
                  case TextOutputCallback.INFORMATION:
                  System.out.println(toc.getMessage());
                  break;
                  case TextOutputCallback.WARNING:
                     System.out.println("WARNING: " +
                        toc.getMessage());
                     break;
                  }
               } else if (callbacks[i] instanceof
                  TextInputCallback) {
                     // prompt the user for information
                     TextInputCallback tic =
                        (TextInputCallback)callbacks[i];
                     // display the prompt like this:
                     //      prompt [default_reply]:
                     System.err.print(tic.getPrompt() +
                                 " [" +
                                 tic.getDefaultText() +
                                 "]: ");
                     System.err.flush();
                     BufferedReader reader =
                        new BufferedReader(
                           new InputStreamReader(System.in));
                        tic.setText(reader.readLine());
               } else if (callbacks[i] instanceof NameCallback) {
```

```
                    ((NameCallback)callbacks[i]).setName(
                       _userName );
                } else if (callbacks[i] instanceof
                    PasswordCallback) {
                       // prompt the user for sensitive information
                       PasswordCallback pc =
                          (PasswordCallback)callbacks[i];
                       System.err.print(pc.getPrompt() + " ");
                       System.err.flush();
                       pc.setPassword(readPassword(System.in));
                } else if (callbacks[i] instanceof
                    LanguageCallback) {
                       // Get the language from the locale
                       LanguageCallback lc =
                          (LanguageCallback)callbacks[i];
                       lc.setLocale( Locale.getDefault() );
                } else {
                    throw new UnsupportedCallbackException
                       (callbacks[i], "Unrecognized Callback");
                }
             }
          }
       /** Reads user password from given input stream. */
       private char[] readPassword(InputStream in) {
          // insert code to read a user password from the
          // input stream
       }
    private String _userName = null;
}
```

Using SASL in the Client

You are ready to authenticate when you have

- determined that there is at least one SASL mechanism in common between the server and your client environment
- implemented `javax.security.auth.callback.CallbackHandler` (if you may need to supply additional credentials during authentication).

The following example shows you how to use SASL in an application:

```
Hashtable props = new Hashtable();
props.put( "javax.security.sasl.client.pkgs",
         "mysecurity.sasl" );
ld.authenticate( dn, props, new SampleCallbackHandler() );
```

Using the External Mechanism

The Netscape Directory SDK for Java includes a mechanism called EXTERNAL. This mechanism verifies that SSL authentication has already completed before it allows a client to connect over LDAP.

To use the EXTERNAL mechanism:

1. Bind to the server and authenticate using SSL.

   For more information, see "Connecting to the Server Over SSL".

2. Call the `LDAPConnection.authenticate` method as follows:

```
ld = new LDAPConnection();
ld.authenticate(null, new String[]{"EXTERNAL"}, null,
                (CallbackHandler)null);
```

`LDAPConnection.authenticate takes the following arguments`:

Table 14.2 Arguments taken by the LDAPConnection.authenticate method

Argument Name	Description
dn	Authentication DN
`mechanisms`	The list of SASL mechanism to use for authentication. If null is specified, the SDK will query the server for all available mechanisms.
`props`	Any optional properties that the mechanism accepts. See Table 14.3 for more details.
`cbh`	An instance of the `CallbackHandler` implemented in your application.

Table 14.3 lists the properties that you can specify for the `props` argument.

Table 14.3 Acceptable values for the props argument.

Property Name	Description
`javax.security.sasl.encryption.minimum`	The minimum key length to be used during the session. The default value is "0" (zero), no session protection. A value of "1" enables integrity protection only.
`javax.security.sasl.encryption.maximum`	The maximum key length to be used during the session. The default value is "256".
`javax.security.sasl.server.authentication`	A boolean value. "True" if a server must authenticate to the client. The default value is "false".
`javax.security.sasl.ip.local`	This is the client's IP address in dotted decimal format. This value is required for KERBEROS_V4 authentication. There is no default value.
`javax.security.sasl.ip.remote`	This is the server's IP address in dotted decimal format. This value is required for KERBEROS_V4 authentication. There is no default value.
`javax.security.sasl.maxbuffer`	Specifies the maximum size of the security layer frames. The default is "0" (zero) meaning that the client will not use the security layer. See
`javax.security.sasl.client.pkgs`	A bar-separated list of package names that are to be used when locating a SaslClientFactory.

The javadocs for the Directory SDK for Java describe the `LDAPConnection` interface and `authenticate` method more fully. For information on using the javadocs, see “Where to Find Reference Information”.

Additional SASL Mechanisms

Authentication using a SASL mechanism other than `EXTERNAL` requires you to implement classes for the mechanism in the client and on the server. For information on obtaining classes for SASL mechanisms see “For More Information”.

For More Information

SASL is described in RFC 2222, which you can find at

`http://www.ietf.org/rfc/rfc2222.txt`

A current listing of registered SASL mechanisms is available at

`http://www.isi.edu/in-notes/iana/assignments/sasl-mechanisms`

Chapter

15

Using the JNDI Service Provider

This chapter explains JNDI and shows you how to use Netscape's LDAP Service Provider for JNDI.

The Java Naming and Directory Interface (JNDI)™ allows Java applications to use a single set of methods to access multiple naming and directory services such as LDAP and NIS. JNDI was developed by the JavaSoft division of Sun Microsystems along with several industry partners, including Netscape.

The chapter contains the following sections:

- "How JNDI Works"
- "Installing the Service Provider"
- "JNDI Environment Properties"
- "Working with Controls"

How JNDI Works

JNDI is a Java API that provides a common way for programmers to access a variety of naming and directory services. The API consists of several packages:

- `javax.naming` for naming operations (access entries)

- `javax.naming.directory` for directory operations (access attributes)
- `javax.naming.event` for requesting event notification
- `javax.naming.ldap` for LDAP-specific features.

JNDI operates through a layer of software called a Service Provider. The Service Provider implements the JNDI operations in terms of a particular underlying protocol.

JNDI's Service Provider Interface (SPI) allows you to select Service Providers at runtime. In many cases you can use the same JNDI methods regardless of whether the Service Provider is talking to an LDAP server or using another protocol such as NIS. In order to access all of the functionality of the LDAP protocol, however, you will have to use methods outside of this abstraction.

For more information about JNDI and Service Providers as well as software, examples, and sample code visit `http://java.sun.com/products/jndi`.

Netscape's LDAP Service Provider

Netscape's LDAP Service Provider is based on version 1.2 of JNDI. In order to have JNDI use Netscape's LDAP Service Provider, you must prepare your environment and select the Service Provider in your code. The following section will help you do this.

Installing the Service Provider

Installing the Netscape LDAP Service Provider involves

- adding the service provider to the `CLASSPATH`
- specifying the Netscape LDAP Service Provider as the JNDI context in your code.
- (Optional) adding the JNDI object schema to the Directory.

The rest of this section will show you how to perform these tasks.

Add the Provider to the Classpath

Before using the Netscape LDAP Service Provider for JNDI in an application launched from the command line, you must add the provider and its associated JAR files to your Java `CLASSPATH`.

To do this, include the following files in your `CLASSPATH`:

- `ldapsp.jar`, the Netscape LDAP Service Provider for JNDI
- `ldapjdk.jar`, the Netscape Directory SDK for Java 4.0 (required by the LDAP Service Provider)
- `jndi.jar`, the Java Naming and Directory Interface version 1.2.

Specify the Service Provider when Creating the Initial Context

In order to communicate with an LDAP server, JNDI needs to know where to find the LDAP Service Provider.

To have it use the Netscape Service Provider, add the following code to your Java application:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.netscape.jndi.ldap.LdapContextFactory");
/* Insert any additional "env.put(... ...)" statements
DirContext ctx = new InitialDirContext(env);
```

Add the JNDI object schema to the Directory (Optional)

Before you can store Java objects in an LDAP directory, you must make sure that the JNDI object schema is available to the directory. If you are using Netscape Directory Server, this requires a modification to your directory schema.

Depending on which version of Netscape Directory Server you are using, there are two ways to do this.

Updating Netscape Directory Server 4.1

If you are using Netscape Directory Server 4.1 you must update the existing JNDI schema file. This is done as follows:

1. Stop the server
2. In the `<server-root>/slapd-<id>/config/` directory, replace the `java-object-schema.conf` file with the one provided in the Netscape Directory SDK for Java.
3. Restart the server

Updating Pre-4.1 Netscape Directory Servers

If you are using Netscape Directory Server 4.0 or earlier, you must add the JNDI schema file. This is done as follows:

1. Stop the server.
2. Copy the `java-object-schema.conf` file that is included with the Netscape Directory SDK for Java to the `<server-root>/slapd-<id>/config` directory.
3. Edit `<server-root>/slapd-<id>/config/ns-schema.conf` to include the following line:

   ```
   include <server-root>/slapd-<id>/config/java-object-schema.conf
   ```

4. Restart the server.

JNDI Environment Properties

JNDI has a number of environment properties that you can set at the system level or pass directly to the initial context via a Hashtable. Netscape's LDAP Service Provider accepts all properties that are defined globally for JNDI. For

those settings that are relevant to LDAP, but not included in the JNDI specification, Netscape's LDAP Service Provider uses the same property names and semantics as the Sun Microsystems LDAP Service Provider.

If a change or addition to the JNDI context environment occurs after an initial context is created, it will be immediately visible unless the changed property pertains to the connection. To force these changes to take effect, invoke the `LdapContext.reconnect()` method. `LdapContext` is an extended `Context` that supports LDAP-specific methods (see "Working with Controls").

The following table contains all JNDI environment properties that are relevant for the Netscape LDAP Service Provider. (Any additional properties that are used by the JNDI Naming Manager are not listed here.) The provider will silently ignore any properties that are not in the table.

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.factory.initial`	Used to select the LDAP provider. To select the Netscape LDAP provider, specify `com.netscape.jndi.ldap.LdapContextFactory`
`java.naming.provider.url`	Specifies LDAP server information. For example: `env.put(Context.PROVIDER_URL, "ldap://dilly.mcom.com:389");` If this property is not set, the Service Provider will attempt to access an LDAP server at port 389 of the local host.
`java.naming.ldap.version`	Specifies the protocol version for the Service Provider. Two values are possible: 2 - selects LDAP Version 2 (LDAPv2) 3 - selects LDAP Version 3 (LDAPv3) For example: `env.put("java.naming.ldap.version", "3");` sets the protocol version to 3. If this property is not set, the Service Provider will attempt to use LDAPv3. If this fails, LDAPv2 is used.

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.security.authentication`	Specifies the authentication mechanism that the Service Provider will use. The following values are permitted: `none` - use no authentication (anonymous) `simple` - use weak authentication (clear text password) If this environment property is not set but the `java.naming.security.principal` property has been set, the Service Provider will use `simple` authentication. If neither property is set, the Service Provider will bind anonymously.
`java.naming.security.principal`	Specifies the DN of the authenticating principal. For example: `env.put(Context.SECURITY_PRINCIPAL, "cn=Directory Manager");` If this property is not set, the Service Provider will bind anonymously.
`java.naming.security.credentials`	Specifies the password of the authenticating principal. For example: `env.put(Context.SECURITY_CREDENTIALS, "secret");`

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.security.protocol`	Specifies the security protocol that the Service Provider will use. One possible value is defined: `ssl` - use Secure Socket Layer This is implemented as follows: `env.put(Context.SECURITY_PROTOCOL, "ssl");` When this property is set and the `java.naming.ldap.factory.socket` property has not been set, the default socket factory `netscape.net.SSLSocket` is used. This class is provided with Netscape Communicator 4.05 and higher.If `java.naming.ldap.factory.socket` has been set, the socket factory specified therein is used.
`java.naming.security.sasl.authorizationId`	Specifies which user DN to use for SASL authentication.
`java.naming.security.sasl.callback`	Specifies a callback handler for SASL mechanisms. This value of this property must be an instance of `javax.security.auth.callback.CallbackHandler`.
`java.naming.security.sasl.client.pkgs`	Specifies a "|"-separated list of packages. These packages are used to located factories that produce SASL mechanism drivers.

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.ldap.factory.socket`	Specifies the class name of a socket factory. This environment property is used to override the default socket factory. For example: `env.put("Java.naming.ldap.factory.socket", "crysec.SSL.SSLSocket");` If the `java.naming.security.protocol` property has been set, but this property is not set, then the default value of `netscape.net.SSLSocket` is used. See Chapter 12, “Connecting Over SSL” for more information.
`java.naming.ldap.ssl.ciphers`	Specifies the suite of ciphers used for SSL connections. These connections are made through sockets created by the factory specified with `java.naming.ldap.factory.socket`. The value of this property is of the type `java.lang.Object`. For example: `env.put("java.naming.ldap.ssl.ciphers", crysec.SSL.SSLParams.getcipherSuite());`
`java.naming.batchsize`	Specifies if searches are to block until all results are available or to return results in batches. A setting of `0` (zero) indicates that the Service Provider should block until all results are received. If this property is not set or is "0" then search results are returned in batches of one.

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.ldap.maxresults`	Specifies the default maximum number of results returned for a search request. `0` (zero) means that there is no limit. If not specified, the default value is `1000`. A request using the parameter `SearchConstraints` in the `DirContext.search()` method can override this value.
`java.naming.referral`	Specifies the maximum number of referrals to follow in a chain of referrals. A setting of `0` (zero) indicates that there is no limit. The default limit is `10`.
`java.naming.ldap.deleteRDN`	Specifies whether the old RDN is removed during `rename()`. If the value is set to `true`, the old RDN is removed. Otherwise, the RDN is not removed. The default value is `true`.
`java.naming.ldap.derefAliases`	Specifies how aliases are dereferenced during search operations. The possible values are: `always` - always dereference aliases `never` - never dereference aliases `finding` - dereference aliases only during name resolution `searching` - dereference aliases only after name resolution. **NOTE**: Netscape Directory Server 3.*x* and 4.*x* do not support aliases.
`java.naming.ldap.typesOnly`	Specifies whether to only return attribute types during searches and calls to `getAttributes()`. Possible values are `true` or `false`. The default is `false`.
`java.naming.ldap.control.connect`	An array of controls to set for an `LDAPConnection when executing LDAP operations`.

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.ldap.attributes.binary`	Specifies attributes that have binary syntax. It extends the Service Provider's list of known binary attributes. The value of this property is a list of comma-separated attribute names. For example: `env.put("java.naming.ldap.attributes.binary", "mpegVideo, mpegAudio");` In contrast to the Netscape Directory SDK for Java, JNDI does not allow you a choice of whether to read attributes as Strings or byte arrays. All attributes are returned as Strings unless they are considered to have binary syntax. The values of attributes that have binary syntax are returned as byte arrays instead of Strings.

Table 15.1 Descriptions of JNDI Environment Properties

Environment Property	Description
`java.naming.ldap.attributes.binary` (continued)	If this property is not set then, by default, only the following attributes and OIDs are recognized as having binary syntax: attribute names containing `;binary` `photo` (`0.9.2342.19200300.100.1.7`) `personalSignature` (`0.9.2342.19200300.100.1.53`) `audio` (`0.9.2342.19200300.100.1.55`) `jpegPhoto` (`0.9.2342.19200300.100.1.60`) `jpegSerialzedData` (`1.3.6.1.4.1.42.2.27.4.1.7`) `thumbnailPhoto` (`1.3.6.1.4.1.1466.101.120.35`) `thumbnailLogo` (`1.3.6.1.4.1.1466.101.120.36`) `userPassword` (`2.5.4.35`) `userCertificate` (`2.5.4.36`) `cACertificate` (`2.5.4.37`) `authorityRevocationList` (`2.5.4.38`) `certificateRevocationList` (`2.5.4.39`) `crossCertificatePair` (`2.5.4.40`) `x500UniqueIdentifier` (`2.5.4.45`)
`java.naming.ldap.ref.separator`	Specifies the character to use when encoding a `RefAddr` object in the `javaReferenceAddress` attribute. This property is used to avoid a conflict should the default separator character appear in the components of a `RefAddr` object. If no value is specified, then the default separator is the hash character (`#`).

Working with Controls

JNDI 1.2 and the Netscape LDAP Service Provider support LDAP controls. Since JNDI only defines a generic interface for controls, the task of defining particular controls and their interfaces is left to the Service Provider. All controls

supported by the Netscape Directory Server are implemented in the `com.netscape.jndi.ldap.controls` package. If you plan to use controls, you will need to import this package in your source code.

Note This is in addition to the JNDI packages that are already specified in your `CLASSPATH`.

Netscape's LDAP Service Provider is implemented on top of the Directory SDK which means that any controls that are available in the SDK are also available through JNDI. These controls use the same API as the Directory SDK, except that they begin with "`Ldap`" instead of "`LDAP`." For instance, the SDK control `LDAPSortControl` is available as `LdapSortControl` in the Netscape LDAP Service Provider for JNDI.

For more information about controls, see Chapter 13, "Working with LDAP Controls."

The following example shows you how to use `LdapSortControl`. Since controls are not part of the generalized directory context (`DirContext`), you must call `getInitialLdapContext()` instead of `getInitialDirContext()`. This creates an `LdapContext` object as the initial context.

```
import java.util.Hashtable;
import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
import com.netscape.jndi.ldap.controls.*;
public class SortReverseOrder {
   public static void main (String[] args) {
      Hashtable env = new Hashtable(5, 0.75f);
      /** Specify the initial context implementation to use.*/
      env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.netscape.jndi.ldap.LdapContextFactory");
      /* Specify host and port to use for directory service */
      env.put(Context.PROVIDER_URL, "ldap://localhost:389");
      LdapContext ctx = null;
      try {
         /* get a handle to an Initial DirContext */
         ctx = new InitialLdapContext(env, null);
         /* specify search constraints to search subtree */
         SearchControls cons = new SearchControls();
         cons.setSearchScope(SearchControls.SUBTREE_SCOPE);
         cons.setReturningAttributes(new String[] { "sn" });
         // specify sort control
         ctx.setRequestControls(
         new Control[] {new LdapSortControl(
            new LdapSortKey[]{
               new LdapSortKey("sn", true, null)},Control.CRITICAL)});
```

```
            /* search for all entries of type "person" */
            NamingEnumeration results = ctx.search("o=mcom.com",
                                        "(objectclass=person)", cons);
            /* for each entry print out name + all attrs and values */
            while (results != null && results.hasMore()) {
               SearchResult si = (SearchResult)results.next();
               Attributes attrs = si.getAttributes();
            /* print each attribute */
            for (NamingEnumeration ae = attrs.getAll();
                  ae.hasMoreElements();) {
               Attribute attr = (Attribute)ae.next();
               String attrId = attr.getID();
               /* print each value */
               for (NamingEnumeration vals = attr.getAll();
                  vals.hasMore();
                  System.out.println(attrId + ": " + vals.next()));}
                  System.out.println();
               }
            }
            catch (NamingException e) {
               System.err.println("Search example failed.");
               e.printStackTrace();
            }
         finally {
            // cleanup
            if (ctx != null) {
               try { ctx.close(); } catch (Exception e) {}
            }
         }
      }
   }
```

Chapter

16

Working with Extended Operations

This chapter explains how LDAP v3 extended operations work and how to use the extended operations that are supported by your LDAP server.

The chapter includes the following sections:

- “How Extended Operations Work”
- “Implementing Support for Extended Operations on the Server”
- “Determining the Extended Operations Supported”
- “Performing an Extended Operation”
- “Example: Extended Operation”

How Extended Operations Work

Extended operations are part of the LDAP v3 protocol. Each extended operation is identified by an OID.

LDAP clients can request the operation by sending an extended operation request. Within the request, the client specifies:

- the OID of the extended operation that should be performed

- data specific to the extended operation

The server receives the request, and performs the extended operation. The server can send back to the client a response containing:

- an OID
- any additional data

In order to use extended operations, both the server and the client must understand the specific extended operation to be performed.

- You need to write a client that can send requests for a specific extended operation and that can receive extended responses from the server.
- Your LDAP server needs to be able to handle requests for specific extended operations and send responses back to the client.

The rest of this chapter describes how to set these up.

Implementing Support for Extended Operations on the Server

If you are running your own Netscape Directory Server 3.0 or later, you can write your own server plug-in that handles extended operations.

You can write an extended operation plug-in that:

- registers the OID of an extended operation as supported (so that the OID appears as a value of the `supportedExtension` attribute; see "Determining the Extended Operations Supported" for more information)
- gets information from an extended operation reqeuest
- creates and sends an extended operation response back to the client

For more information, see the *Netscape Directory Server Plug-In Programmer's Guide.*

Determining the Extended Operations Supported

To determine the extended operations supported by the server, get the root DSE of the server, and check the `supportedExtension` attribute. The values of this attribute are the object identifiers (OIDs) of the extended operations supported by this server.

If the root DSE does not have a supportedExtension attribute, the server does not support any extended operations.

For information on getting the root DSE, see “Getting the Root DSE”.

Performing an Extended Operation

To perform an extended operation, do the following:

1. Construct a new `LDAPExtendedOperation` object, specifying the OID of the extended operation and the data that you want applied to the operation.
2. Invoke the `extendedOperation` method of the `LDAPConnection` object, passing it the newly constructed `LDAPExtendedOperation` object.

The `LDAPExtendedOperation` object that this method returns represents the server’s response. You can invoke the `getID` and `getValue` methods of this object to get the OID and the data from the server’s response.

Example: Extended Operation

The following program is an example of an LDAP client that request an extended operation with the OID 1.2.3.4 from the server.

```
...
import netscape.ldap.*;
import java.util.*;
import java.io.*;
...
public class ReqExtOp {
```

```
public static void main( String[] args ) {
   LDAPConnection ld = null;
   int status = -1;
   try {
      ld = new LDAPConnection();

      /* Connect to server */
      String MY_HOST = "localhost";
      int MY_PORT = LDAPv2.DEFAULT_PORT;
      ld.connect( MY_HOST, MY_PORT );
      System.out.println( "Connected to server." );

      /* Authenticate to the server as directory manager */
      String MGR_DN = "cn=Directory Manager";
      String MGR_PW = "23skidoo";
      if ( ld.LDAP_VERSION < 3 ) {
         ld.authenticate( 3, MGR_DN, MGR_PW );
      } else {
         System.out.println( "Specified LDAP server does not " +
            "support v3 of the LDAP protocol." );
         ld.disconnect();
         System.exit(1);
      }
      System.out.println( "Authenticated to directory." );

      /* Create an extended operation object */
      String myval = "My Value";
      byte vals[] = myval.getBytes( "UTF8" );
      LDAPExtendedOperation exop =
         new LDAPExtendedOperation("1.2.3.4", vals );
      System.out.println( "Created LDAPExtendedOperation object." );

      /* Request the extended operation from the server. */
      LDAPExtendedOperation exres = ld.extendedOperation( exop );
      System.out.println( "Performed extended operation." );

      /* Get data from the response sent by the server. */
      System.out.println( "OID: " + exres.getID() );
      String retValue = new String( exres.getValue(), "UTF8" );
      System.out.println( "Value: " + retValue );
   }
   catch( LDAPException e ) {
      System.out.println( "Error: " + e.toString() );
   }
   catch( UnsupportedEncodingException e ) {
      System.out.println( "Error: UTF8 not supported" );
   }

   /* Done, so disconnect */
   if ( (ld != null) && ld.isConnected() ) {
```

```
            try {
               ld.disconnect();
            } catch ( LDAPException e ) {
               System.out.println( "Error: " + e.toString() );
            }
         }
         System.exit(status);
      }
   }
   ...
```

Chapter

17

Using the Asynchronous Interface

This chapter shows you how to use the Asynchronous Interface to LDAP in Java applications.

This chapter contains the following sections:

- "Synchronous vs. Asynchronous Connections"
- "Common Uses for the Asynchronous Interface"
- "New Classes in the Asynchronous Interface"
- "Performing Asynchronous Searches"
- "Where to Go for More Information"

Synchronous vs. Asynchronous Connections

Most operations using the Directory SDK for Java are performed synchronously. A connection is established, a request is sent, the results are returned, and the application resumes. Though the SDK can deliver one search result at a time, other operations block until completion when accessing an LDAP server.

Sometimes it is useful to initiate a new request while another one executes. An additional interface is provided to access the SDK's built-in support for these asynchronous requests. By returning control to an application before obtaining a response, the Asynchronous Interface allows you to perform complex operations requiring access to low-level LDAP mechanisms.

Beginning with version 4.0 of the Directory SDK for Java, `LDAPConnection` methods support both asynchronous and synchronous requests. Synchronous methods wait for response messages from a server and then process them for you. Asynchronous methods require you to check for the messages and perform the processing in your code. This allows you to make additional LDAP requests while waiting for results to return.

Common Uses for the Asynchronous Interface

Since it involves managing more complex code in an application, it is best to use the asynchronous methods only when required. The most common use is for merging the results of searches that involve multiple servers or that are executed simultaneously on different subtrees. This is sometimes referred to as "multiplexing."

A search that multiplexes servers can make a request to an array of hosts. A search that multiplexes query statements can make different requests to different subtrees of a server. If you combine these search methods you can perform complex searches across a number of servers without having to wait for individual responses.

The following example illustrates a practical use of multiplexed searches and the asynchronous interface.

Suppose we want to implement event notification as a generic service using LDAP persistent search. Synchronous methods require a new thread for every request to the service. This solution is not scalable and can exhaust system resources very quickly.

If we rewrite the search using the asynchronous interface, performance will improve dramatically. Since asynchronous searches do not block until completion, we can multiplex the persistent search results into one queue and then process them on a single thread.

New Classes in the Asynchronous Interface

The Directory SDK for Java handles asynchronous communication through the `LDAPAsynchronousConnection` interface and its dependent classes. These files collectively form the asynchronous extensions to the LDAP API.

`LDAPAsynchronousConnection` defines methods for authenticating to a server, as well as for searching, modifying, comparing and deleting entries in the directory.

When you call a method of `LDAPAsynchronousConnection`, it returns a listener object. This object acts as a message queue and accepts search results and server-generated responses to LDAP requests. It is the responsibility of the LDAP client to read and process these messages.

`LDAPAsynchronousConnection` incorporates the following classes which handle asynchronous client-server interactions:

- `LDAPMessage`, which is the base class for LDAP request and response messages.
- `LDAPResponse`, which extends `LDAPMessage`, represents a message received from an LDAP server in response to a request.
- `LDAPExtendedResponse`, which extends `LDAPResponse`. It is the response that an LDAP server returns when handling an extended operation request.
- `LDAPResponseListener` queues `LDAPResponse` messages.
- `LDAPSearchResult`, which extends `LDAPMessage`. It contains a single LDAP entry and is one of the responses that an LDAP server can return when handling a search request.
- `LDAPSearchResultReference`, which extends `LDAPMessage`. It contains a referral and is one of the responses that an LDAP server can return when handling a search request.
- `LDAPSearchListener` queues search results and references.

The rest of this chapter shows you how to use the asynchronous interface to perform multiplexed searches.

Performing Asynchronous Searches

One of the most common uses of the asynchronous interface is for performing multiplexed searches using more than one server or query. The rest of this section will show you how to do this.

Searching Multiple Servers

To perform a search on more than one server:

1. Connect to all the servers.
2. Create a response listener for one search.
3. Share the response listener all the other searches.
4. Obtain and process the results.
5. Disconnect from the servers.

The following code demonstrates how to do this in an application:

```
import netscape.ldap.*;
import java.util.*;
/* This example multiplexes the input from three different servers */
public class MultiplexServers {
   public static void main( String[] args )
   {
      try {
         LDAPAsynch[] ld = new LDAPAsynch[3];
         String[] hosts = { "foo1", "foo2", "foo3" };
         int[] ports = { 389, 389, 2018 }
         String[] bases =
            { "o=Airius.com", "o=Acme.com", "dc=Acme,dc=com" };
         /* search for all entries with surname of Jensen */
         String MY_FILTER = "sn=Jensen";
         for( int i = 0; i < ld.length; i++ ) {
            ld[i] = new LDAPAsynch();
            /* Connect to server */
            ld[i].connect( hosts[i], ports[i] );
         }
         /* Get a response listener for one search */
         LDAPSearchListener l =
            ld[0].search( bases[0],
                          ld.SCOPE_SUB,
```

```
                    MY_FILTER,
                    null,
                    false,
                    (LDAPSearchListener)null );
   /* Share the listener */
   for( i = 1; i < ld.length; i++ ) {
      ld[i].search( bases[i],
                    ld[i].SCOPE_SUB,
                    MY_FILTER,
                    null,
                    false,
                    l );
   }
   /* Loop on results until finished */
   LDAPMessage msg;
   while( (msg = l.getResponse()) != null ) {
      if ( msg instanceof LDAPSearchResultReference ) {
         String[] urls =
            ((LDAPSearchResultReference)msg).getUrls();
         // Do something with the referrals...
      } else if ( msg instanceof LDAPSearchResult ) {
         LDAPEntry entry =
            ((LDAPSearchResult)msg).getEntry();
         // The rest of the processing is the same as for
         // a synchronous search
         System.out.println( entry.getDN() );
      } else {
         // A search response
         LDAPResponse res = (LDAPResponse)msg;
         int status = res.getResultCode();
         if ( status == LDAPException.SUCCESS ) {
            // Nothing to do
         } else {
            String err =
               LDAPException.errorCodeToString(status);
            throw new LDAPException(
                                    err,
                                    status,
                                    res.getErrorMessage(),
                                    res.getMatchedDN() );
         }
      }
   }
} catch ( LDAPException e ) {
   System.err.println( e.toString() );
}
/* Done, so disconnect */
if ( ld.isConnected() ) {
   ld.disconnect();
}
```

```
   }
}
```

Multiple Search Statements

To perform multiple searches in different subtrees of a single server:

1. Connect to the server.
2. Create a response listener for one search.
3. Share (multiplex) the response listener with the other searches.
4. Obtain and process the results.
5. Disconnect from the server.

The following code demonstrates how to do this in an application:

```
import netscape.ldap.*;
import java.util.*;
/* This example multiplexes the input from three searches in
different subtrees of the same server */
public class MultiplexTrees {
   public static void main( String[] args )
   {
      try {
         LDAPAsynch ld = new LDAPAsynch();
         /* Connect to server */
         String MY_HOST = "localhost";
         int MY_PORT = 389;
         ld.connect( MY_HOST, MY_PORT );
         String MY_FILTER = "sn=Jensen";
         String[] bases =
            { "o=Airius.com", "o=Acme.com", "dc=Acme,dc=com" };
         /* Get a response listener for one search */
         LDAPSearchListener l =
            ld.search( bases[0],
                       ld.SCOPE_SUB,
                       MY_FILTER,
                       null,
                       false,
                       (LDAPSearchListener)null );
         /* Share the listener */
         for( i = 1; i < bases.length; i++ ) {
            ld.search( bases[i],
                       ld.SCOPE_SUB,
```

```
                MY_FILTER,
                null,
                false,
                l );
}
/* Loop on results until finished */
LDAPMessage msg;
while( (msg = l.getResponse()) != null ) {
   /* The rest is the same as in the previous example */
   /* ... */
```

Where to Go for More Information

The javadocs for the Directory SDK for Java describe all the classes, methods and exceptions of the `LDAPAsynchronousConnection` interface. For more information on using the javadocs, see “Where to Find Reference Information”.

The Internet-Draft titled “The Java LDAP Application Program Interface Asynchronous Extension” is available at the following URL:

`http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-java-api-asynch-ext-00.txt`.

Note Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

Glossary

This glossary defines terms commonly used when working with LDAP.

base DN The distinguished name (DN) that identifies the starting point of a search.

For example, if you want to search all of the entries in the "ou=People,o=Airius.com" subtree of the directory, "ou=People,o=Airius.com" is the base DN.

For more information on base DNs and searching the directory, see "Specifying the Base DN and Scope".

continuation reference See search reference.

control LDAP controls are specified as part of the LDAP v3 protocol. A control provides the means to specify additional information for an operation. Clients and servers can send controls as part of the requests and responses for an operation.

For more information on LDAP controls, see Chapter 13, "Working with LDAP Controls".

DIT The hierarchical organization of entries that make up a directory. DIT stands for "Directory Information Tree."

DSA An X.500 term for an LDAP server. DSA stands for "Directory System Agent."

DSE An entry containing server-specific information. DSE stands for "DSA-specific entry." Each server has different attribute values for the DSE.

extended operation An extension mechanism in the LDAP v3 protocol. You can define extended operations to perform services not covered by the protocol. The extended operation mechanism specifies the means for an LDAP client to request a custom operation (not specified in the LDAP protocol) from an LDAP server.

For more information on extended operations, see Chapter 16, "Working with Extended Operations".

LDIF LDAP Data Interchange Format. The format is specified in the Internet-Draft "The LDAP Data Interchange Format -- Technical Specification," which is available at the following location: `http://www.ietf.org/internet-drafts/draft-good-ldap-ldif-04.txt`. Note that Internet-Drafts expire every six months. If the URL above does not work, try incrementing the number by one. For example, `draft-06.txt` would become `draft-07.txt`.

operational attributes Attributes that are used by servers for administering the directory. For example, creatorsName is an operational attribute that specifies the DN of the user who added the entry. Operational attributes are not returned in any search results unless you specify the attribute by name in the search request.

For more information on searching the operational attributes, see "Specifying the Attributes to Retrieve".

referral Refers an LDAP client to another LDAP server. An LDAP server can be configured to send your client a referral if your client requests a DN with a suffix that is not in the server's directory tree (for example, if the directory includes entries under "o=Airius.com" and your client requests an entry under "o=AiriusWest.com").

Referrals contain LDAP URLs that specify the host, port, and base DN of another LDAP server.

Note that referrals are not the same as (but are similar to) search references. A search reference is returned as part of the results of a search; a referral is returned when the base DN of a search (or the target DN of any other LDAP operation) is not part of the LDAP server's directory tree.

For more information on handling referrals, see "Handling Referrals".

referral hop limit The maximum number of referrals that your client should follow in a row. For example, suppose your client receives a referral from LDAP server A to LDAP server B. After your client follows the referral to LDAP server B, that server

sends you a referral to LDAP server C, which in turn refers you to LDAP server D. Your client has been referred 3 times in a row. If the referral hop limit is 2, the referral hop limit has been exceeded.

For more information on handling referrals, see “Handling Referrals”.

root DSE An entry (a DSE) that is located at the root of the DIT.

For information on getting the root DSE of an LDAP server, see “Getting the Root DSE”.

search reference Also known as continuation references, search result references, or smart referrals. A search reference is an entry in the directory that refers to another LDAP server (the reference is in the form of an LDAP URL).

Search references are returned in search results along with entries found in the search. (A referral, on the other hand, is returned before searching through any entries. A referral is returned if the base DN does not have a suffix that is handled by the server.)

For more information on handling search references, see “Getting the Search Results”.

search result reference See search reference.

server plug-in Beginning with version 3.0, Netscape Directory Server supports a plug-in interface that allows you to extend the functionality of the server. You can write plug-ins that handle extended operations or SASL authentication requests. For more information on server plug-ins, see the Netscape Directory Server Programmer’s Guide.

smart referral See search reference.

subschema entry Entry containing all the schema definitions (definitions of object classes, attributes, matching rules, and so on) used by entries in part of a directory tree.

For more information on getting the subschema entry, see “Getting the Schema for an LDAP Server”.

Index

D

E

F

I

J

L

M

N

O

P

R

S

U