Programmer's Guide

Netscape LDAP SDK for C

Version 4.1

November 20, 2000

Contents

Preface

The *LDAP SDK for C Programmer's Guide* documents the Netscape LDAP SDK for C, a software development kit (SDK) for writing Lightweight Directory Access Protocol (LDAP) client applications. This SDK supports the *The C LDAP Application Program Interface Internet-Draft*. The draft is a formal description of the C language API for LDAP.

The Netscape LDAP SDK for C includes the C libraries for the LDAP application programming interface (API). You use the functions contained in this API to enable your applications to connect to, search, and update LDAP servers located on a network or on the Internet. For a more detailed explanation of LDAP, see Chapter 2, "An Introduction to LDAP."

This Preface contains the following sections:

- What You Are Expected to Know
- About This Release
- Organization of This Guide
- Documentation Conventions
- Where to Find Additional Information

What You Are Expected to Know

This guide is intended for use by C and C++ programmers who want to enable new or existing client applications to connect to LDAP servers. The functionality contained in the LDAP SDK for C allows you to search and update databases that are managed by LDAP servers. This book assumes that you are familiar with writing and compiling C applications on the platform(s) that you want to implement your client applications.

About This Release

The Netscape LDAP SDK for C, v4.1 has been updated to provide even more support for LDAPv3 than the previous 4.0 release. For details on the new features, refer to the Release Notes that accompany the SDK and the `ldap.h` header file.

While this manual has been updated to cover many of the new features contained this release, there are several areas of functionality that have not been fully documented. In particular, documentation is forthcoming for the Netscape UTF8 extensions and the LDAPv3 extended I/O callback interface.

Deprecated Functions

Although several functions have been deprecated in this release of the LDAP SDK for C, the SDK still supports these deprecated functions for backward compatibility. The following table outlines some of the deprecated functions and the functions you should use in their place whenever you write new code. For more information on the functions that have been deprecated, refer to the `ldap.h` header file and the most recent C LDAP API Internet Draft.

Table 1 Deprecated functions and their suggested replacements

Deprecated Function	Suggested Replacement Function(s)
`ldap_bind()`	`ldap_simple_bind()` or `ldap_sasl_bind()`
`ldap_bind_s()`	`ldap_simple_bind_s()` or `ldap_sasl_bind_s()`
`ldap_build_filter()`	`ldap_create_filter()`
`ldap_cache_flush()`	`ldap_memcache_*()` functions
`ldap_modrdn` `ldap_modrdn2()`	`ldap_rename()`
`ldap_modrdn_s()` `ldap_modrdn2_s()`	`ldap_rename_s()`
`ldap_open()`	`ldap_init()`
`ldap_perror()`	`ldap_get_lderrno()` (to get the result code) and `ldap_err2string()` (to get the corresponding error message for this result code)
`ldap_result2error()`	`ldap_parse_result()`
`ldap_setfilteraffixes()`	`ldap_set_filter_additions()`
`ldap_version`	`ldap_get_option()` with the `LDAP_OPT_API_INFO` option

Organization of This Guide

This guide explains how to use the Netscape LDAP SDK for C to enable applications to interact with LDAP servers. The guide documents the LDAP API, which consists of data structures and functions used to communicate with LDAP servers.

This manual is divided into four parts:

- Part 1, "Getting Started," explains how you can use the LDAP SDK for C to enable your applications for LDAP.
- Part 2, "Writing Clients with the Netscape LDAP SDK for C," explains how you can use the LDAP SDK for C to enable your applications for LDAP communications.
- Part 3, "Advanced Topics," contains additional material that you might need, including descriptions of entries and attributes.
- Part 4, "Reference," describes each data structure, function, and status code in the LDAP API.

Table 2 details the chapters contained in the book:

Table 2 Finding information in this manual

Chapter	Description
Preface	This chapter
Part 1, "Getting Started"	
Chapter 1, "The Netscape LDAP SDK for C"	Understand the components of the Netscape LDAP SDK for C and how to compile your own applications.
Chapter 2, "An Introduction to LDAP"	Learn more about LDAP and the Directory Server.
Chapter 3, "A Short Example"	See the complete source code for a sample client.
Part 2, "Writing Clients with the Netscape LDAP SDK for C"	
Chapter 4, "Writing an LDAP Client"	Write a program that connects, searches, and updates an LDAP server.
Chapter 5, "Using the LDAP API"	Handle errors, understand synchronous and asynchronous functions, handle LDAP referrals.
Chapter 6, "Searching the Directory"	Search an LDAP server for entries.

Table 2 Finding information in this manual

Chapter	Description
Chapter 7, “Using Filter Configuration Files”	Create a search filter or parse a filter configuration file.
Chapter 8, “Adding, Updating, and Deleting Entries”	Add, update, or remove a directory entry from an LDAP server.
Chapter 9, “Comparing Values in Entries”	Compare a value against the attribute values in a directory entry.
Chapter 10, “Working with LDAP URLs”	Interpret an LDAP URL.
Part 3, “Advanced Topics”	
Chapter 11, “Getting Server Information”	Get information about the server over the LDAP protocol.
Chapter 12, “Connecting Over SSL”	Connect to an LDAP server over SSL.
Chapter 13, “Using SASL Authentication”	Use a SASL mechanism to authenticate to the LDAP server.
Chapter 14, “Working with LDAP Controls”	Send, receive, and interpret LDAPv3 controls.
Chapter 15, “Working with Extended Operations”	Perform LDAPv3 extended operations.
Chapter 16, “Writing Multithreaded Clients”	Write a multi-threaded client.
Part 4, “Reference”	
Chapter 17, “Data Types and Structures”	Look up the description of a data structure.
Chapter 18, “Function Reference”	Look up the description of a function.
Chapter 19, “Result Codes”	Look up the description of a status code returned by an API function.

Documentation Conventions

This book uses the following font conventions:

- `Monospace type` is used for all sample code and code listings.

- `Monospace type` is also used within paragraph text to represent language elements (such as function names and class names), reserved names, filenames, pathnames, directory names, HTML tags, and any text that must be typed at a terminal.
- *Italic serif font* is used within code and language elements to indicate variable placeholders. For example, in the following command, *filename* is a variable placeholder:

  `gunzip` *filename*`.tar.gz`
- *Italic type* is used within paragraph text for book titles, emphasis, variables, and placeholders.

Where to Find Additional Information

Netscape and iPlanet provide binary releases of this SDK. However, note that this SDK is also available in source code form as part of the *Mozilla.org* open source project. Refer to the following site for more information on how you can get the source code and contribute to the further development of this SDK:

`http://www.mozilla.org/directory`

Related Information

The following table lists several sources of information on LDAP and its associated technologies. Note that some of the links in this table are time-sensitve and the drafts might expire.

Table 3 Sources of information on LDAP and associated protocols

RFC or Internet-Draft / Internet Location
Lightweight Directory Access Protocol (v2), RFC 1777 `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1777.txt`
Lightweight Directory Access Protocol (v3), RFC 2251 `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2251.txt`
The C LDAP Application Program Interface, Internet-Draft Revision 5 `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-c-api-05.txt`
LDAP Control Extension for Server Side Sorting of Search Results, RFC 2891 `ftp://ftp.isi.edu/in-notes/rfc2891.txt`
Simple Authentication and Security Layer (SASL), RFC 2222 `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2222.txt`

Table 3 Sources of information on LDAP and associated protocols

RFC or Internet-Draft / Internet Location
Secure Sockets Layer (SSL) Protocol (v3), Internet-Draft Revision 3 `http://home.netscape.com/eng/ssl3/ssl-toc.html`
Access Control Model for LDAPv3, Internet-Draft Revision 6 `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-acl-model-06.txt`
LDAP Extensions for Scrolling View Browsing of Search Results, Internet-Draft Revision 4 `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldapv3-vlv-04.txt`
LDAP Control Extension for Server Side Sorting of Search Results `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-sorting-00.txt`
LDAP Proxied Authorization Control, Revision 5 `http://www.ietf.org/internet-drafts/draft-weltman-ldapv3-proxy-05.txt`
LDAP Authentication Response Control, Internet-Draft Revision 1 `http://www.ietf.org/internet-drafts/draft-weltman-ldapv3-auth-response-01.txt`
Persistent Search: A Simple LDAP Change Notification Mechanism, Internet-Draft Revision 2 `http://www.ietf.org/internet-drafts/draft-ietf-ldapext-psearch-02.txt`
Netscape Portable Runtime Library `http://www.mozilla.org/projects/nspr`

iPlanet Technical Documentation

In addition to these sources of information, you can find addition information on the LDAP protocol, the iPlanet Directory Server, and its associated technologies at the following iPlanet development site:

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

You can also find information on the LDAP protocol in the documentation for the iPlanet Directory Server. These documents, along with the installation instructions, release notes, and documentation for all iPlanet and Netscape servers, can be found at the following location:

`http://docs.iplanet.com/docs/manuals/index.html`

Part 1

Getting Started

Chapter 1

The Netscape LDAP SDK for C

This chapter introduces the Lightweight Directory Access Protocol (LDAP) Application Programming Interface (API) and describes the Netscape LDAP SDK for C.

This chapter contains the following sections:

- The LDAP API
- Files Provided With the LDAP SDK for C
- Compiling LDAP Clients
- Running LDAP Application Programs
- The Example Programs

The LDAP API

The *Lightweight Directory Access Protocol (v3)*, RFC 2251 defines a set of API functions that you can use to build LDAP-enabled clients. The functionality implemented in this SDK closely follows the interfaces outlined in RFC 2251. Using the functionality provided with this SDK, you can enable your clients to connect to LDAPv3-compliant servers and perform standard LDAP functions. Among other things, with this SDK you can:

- Search for and retrieving a list of entries.
- Ad new entries (modify) the database.
- Update existing entries.
- Delete entries.
- Rename entries.

For example, if you are writing an email application, you can use the functions in the LDAP API to retrieve email addresses from an LDAP server.

The API functions allow you to perform LDAP operations synchronously or asynchronously.

- You can call a synchronous function if you want to wait for the operation to complete before receiving the return value of a function.
- If you want to perform other work while waiting for an operation to complete, you can call an asynchronous function and poll the LDAP client library to check the status of the operation.

The LDAP API also includes functionality defined in LDAPv3. For example, you can call functions to request extended operations from an LDAPv3 server.

Subsequent chapters in this manual explain how to use the LDAP API functions contained in the LDAP SDK for C. For a complete list of the functions implemented in this SDK, refer to Chapter 18, “Function Reference.”

Files Provided With the LDAP SDK for C

The Netscape LDAP SDK for C is a Software Development Kit (SDK) that contains C header files, C libraries, tools, and example programs. To install the SKD, you download the compressed SDK package from the iPlanet web site on the Internet, and unpack the files to the directory of your choice. When you unpack the SDK, it will create several directories and populate them with the various files provided with the SDK. The directories and files supplied with the SDK are detailed in the sections below.

Directories Installed With the SDK

The LDAP SDK for C installation process creates several directories. These directories are populated with the files provided by the SDK. The table below outlines the directories that are created by the installation process:

Table 1-1 LDAP SDK for C directories

Directory	Description
`include`	This directory contains the header files for the LDAP SDK for C. In your client source files, you must include the files contained in this directory as described in the section “Include Files Supplied with the LDAP SDK for C.”

Table 1-1 LDAP SDK for C directories

Directory	Description
`lib`	This directory contains the C library files provided with the LDAP SDK for C. For details on these files, see the section "Libraries Supplied with the LDAP SDK for C."
`examples`	This directory contains sample source code and makefiles for LDAP clients. See the `README` file in this directory for information about building these client programs.
`tools`	This directory contains the LDAP command-line tools. These are similar to the tools provided with the iPlanet Directory Server and are described in the iPlanet Directory Server *Command and File Reference* (see `http://docs.iplanet.com/docs/manuals/directory/` for details). Note that in order to use these applications, you need to make sure that the application can find the LDAP API shared library or dynamic link library, as described in the section "Libraries Supplied with the LDAP SDK for C."
`docs`	Initially this directory contains only the README file that is shipped with this version of the SDK. However, the directory is intended as a location for this guide (the *LDAP SDK for C Programmer's Guide*) and the associated product Release Notes, which you can download from the iPlanet documentation web site detailed in the README file.

Include Files Supplied with the LDAP SDK for C

The Netscape LDAP SDK for C ships with the following header files:

Table 1-2 Header files shipped with the LDAP SDK for C

Header File	Description
`lber.h`	Prototypes the standard Basic Encoding Rules (BER) functions, structures and defines for the LDAP SDK.
`ldap.h`	Prototypes for the standard functions, structures, and defines contained in the LDAP SDK.
`ldap_ssl.h`	Prototypes for the SSL functions, structures, and defines contained in the LDAP SDK.
`ldappr.h`	Prototypes the functions, structures, and defines contained in the Netscape Portable Runtime (NSPR) for the LDAP SDK.

Libraries Supplied with the LDAP SDK for C

The Netscape LDAP SDK for C comes with several different library sets. The specific library files that you link with depend on the type of application(s) you are building. Table 1-3 outlines the library files provided with the Netscape LDAP SDK for C, v4.1:

Table 1-3 Library files shipped with the LDAP SDK for C

Library File	Description
`libldapssl41.so`	The SSL-enabled library for Solaris, AIX, and Linux.
`libnspr3.so`	Library containing the core Netscape Portable Runtime (NSPR) functions.
`libldapssl41.sl`	The SSL-enabled library for HP-UX.
`libnspr3.sl`	Library containing the core NSPR functions for HP-UX.
`nsldapssl32v41.dll`	The SSL-enabled DLL library for 32-bit Microsoft Windows.
`nsldapssl32v41.lib`	The SSL-enabled import library for 32-bit Microsoft Windows.
`libnspr3.dll`	Library containing the core NSPR functions, utilizes Microsoft fibers (specific to Windows NT and Windows 2000).
`nspr3.dll`	Library containing the core NSPR functions.
`nsldap32v41bc.lib`	The standard import library for 32-bit Microsoft Windows, to be used with Inprise (Borland) compilers. This library is not is not officially supported at this time.

Tools Supplied With the LDAP SDK for C

The LDAP SDK for C includes the following utilities that help you to work with LDAP data sets:

Table 1-4 Utilities supplied with the LDAP SDK for C

Utility	Description
`ldapmodify`	Lets you modify, add, delete, or rename directory entries.
`ldapsearch`	Lets you search for directory entries and display attributes and values found.
`ldapdelete`	Lets you delete existing directory entries.
`ldapcmp`	Lets you compare entries between directory servers.

A list of options for each utility can be obtained by typing the utility name at the command line. For detailed documentation on these utilities, refer to the *Command and File Reference* shipped with the iPlanet Directory Server.

Compiling LDAP Clients

The Netscape LDAP SDK for C includes the header files and libraries for the LDAP API functions. You can include these header files and link to these libraries to enable your application to use LDAP.

Including the LDAP Header File

To make use of the functions contained in the LDAP SDK, whether you are developing on UNIX or in a 32-bit Windows system, you must include the `ldap.h` header file in your C source files. The following line of code shows as shown in the following line of code:

```
#include “ldap.h”
```

The `ldap.h` header file declares the functions and structures contained in the LDAP API.

You do not need to explicitly include the `lber.h` header file; `lber.h` is already included in `ldap.h`.

If you are calling the LDAP SSL functions, you also need to include the `ldap_ssl.h` header file, as follows:

```
#include “ldap_ssl.h”
```

To make use of the Netscape Portable Runtime (NSPR) with your LDAP applications, you must include `ldappr.h` (this header file contains the prototypes for functions that tie the LDAP libraries to NSPR). You can find more information on NSPR in the `ldappr.h` header file and at the following URL:

```
http://www.mozilla.org/projects/nspr
```

Compiling Clients on UNIX

The Netscape LDAP SDK for C has an `include` directory that contains C header files and a `lib` directory that contains the shared libraries and objects to which you must link.

When compiling clients on UNIX platforms, you need to link to the appropriate LDAP API shared library. On Solaris, AIX, Linux, and OSF, the shared library is `libldapssl41.so`. On HP-UX, the library is named `libldapssl41.sl`. For example, on Solaris, you would use the following link option:

```
-L../lib -lldapssl41
```

Refer to the `Makefile` in the `examples` directory for details on compiling your applications.

NOTE The NSPR libraries are now distributed with the LDAP SDK for C. When linking with the SSL-enabled applications, you must also link with the NSPR libraries. For more information on the NSPR libraries, refer to the documentation at the following URL (follow the "Documentation and Training" link):
`http://www.mozilla.org/projects/nspr/`

If you linked the shared library, you need to make sure that the LDAP client can find the library. Do one of the following:

- Make sure that the LDAP API shared library file (such as `libldap41.so` for Solaris) is in a standard location, as in `/usr/lib`. Alternatively, you can use environment variables to specify the library location.
- On some platforms, if you have compiled your client with certain flags set, you can set an environment variable to specify the run-time path to check when loading libraries. If you do this, you can set this variable to the location of the LDAP library.

  For example, on Solaris and Linux, you can use the `LD_LIBRARY_PATH` environment variable. On HP-UX, you can use the `SHLIB_PATH` environment variable if you use the `-Wl,+s+b` flag when compiling and linking. On AIX, you can use the `LIBPATH` environment variable.
- Use a link flag that specifies the path where the executable can find the library. For example, on Solaris, you can use the `-R` flag to specify the path where the executable can find the library.

See the makefile in the `examples` directory for examples of additional flags and defines that you might need to specify when compiling and linking your LDAP client. Different platforms might require different sets of define statements.

Compiling Clients on 32-bit Windows Systems

When compiling clients on 32-bit Windows systems (Windows NT, Windows 95, Windows 98, and Windows ME), make sure to define `_CONSOLE` if you are writing a console application, or `_WINDOWS` if you are writing a standard Windows GUI application.

Make sure to link to the LDAP API import library. The name of the library is `nsldapssl32v41.lib` for Microsoft compilers.

Make sure to copy the `nsldap32v41.dll` dynamic link library to a directory where your client can find it.

NOTE Some iPlanet servers (such as the iPlanet Directory Server) might install a different version of the `nsldap32v41.dll` in the Windows `system32` directory. Make sure that your client finds the version of the DLL that is included with the Netscape LDAP SDK for C before finding the version in the `system32` directory. For example, you can copy the SDK DLL to the same directory as your client application.

During run-time, your client searches for the LDAP API dynamic link library in the following locations:

1. The directory from which the application loaded.
2. The current directory.
3. The 32-bit Windows system directory, typically `winnt\system32`.

   Note that a version of the LDAP API DLL may already exist in this directory. For example, other iPlanet servers might install a different version of this DLL. To avoid potential conflicts, you should not install the DLL in this directory.
4. The Windows directory.
5. The directories listed in the `PATH` environment variable.

Running LDAP Application Programs

When you run LDAP clients on UNIX or Windows system, you must ensure that the operating system can find the libraries that support the LDAP functions that are called by your application. On UNIX, these files are called shared objects; on Windows, they are call Dynamic Link Libraries. Generally, these files are referred to as runtime libraries.

The UNIX Runtime Library Files

On Unix systems (such as Solaris, AIX, and HP-UX), the library files consist of a shared object library. When running clients built with these libraries, make sure to set the appropriate environment variable that specifies the run-time path used to find libraries.

For example, on Solaris and Linux, set the `LD_LIBRARY_PATH` environment variable to the location of the shared library. On HP-UX, set the `SHLIB_PATH` environment variable if you are using the `-Wl,+s+b` flag to compile your client application. On AIX, set the `LIBPATH` environment variable.

The Windows Runtime Library Files

For Windows applications, the library files include an import library and a dynamic link library. When running clients built with these libraries, make sure to copy the dynamic link library to the directory containing your client or copy it to a location where the DLL can be found by the operating system.

The Example Programs

The Netscape LDAP SDK for C ships with several examples that demonstrate the use of the functions contained in the LDAP SDK for C library. This code was tested on a Solaris 2.6 machine and on a Windows NT 4.0 machine with Microsoft Visual C++ 6.0 SP3.

Located in the `examples` directory of the installed the SDK, the example code assumes that you are running clients against a directory server that is compliant with LDAPv3 (such as the Netscape Directory Server, version 4.12) and that you have the sample database (`airius.com`) loaded on the server. (If you want to run the examples, you will need to have a working LDAP server running and the sample data contained in `airius.com` must be properly loaded.)

To compile and run these examples, refer to the `README` file contained in the `examples` directory. The Windows NT versions of the LDAP SDK for C include sample project files. Sample Visual C++ makefiles for a 32-bit Windows application (winldap) are included in the `windows` example directory. The Win32 version of the makefile is named `winldap.mak`.

Synchronous Examples

These samples use the synchronous LDAP calls. These function calls are more straightforward to use than their asynchronous counterparts. Because of this, it is suggested you look at these examples first.

The synchronous calls block the calling process until all results have been returned, so they are probably not appropriate for use with clients that implement a graphical user interface (GUI) in a single-threaded environment because these programs usually rely on event loops. However, these sample programs fine for command-line clients and CGI programs.

Table 1-5 Synchronous example programs

Example Program	Description
`search.c`	Shows how to use `ldap_search_s()` to search for all entries which have an attribute value which exactly matches what you're searching for. In this example, all entries with the surname (last name) "`Jensen`" are retrieved and displayed.
`csearch.c`	Like `search.c`, but enables an in-memory cache.
`ssnoauth.c`	Like `search.c`, but the search is done over an SSL protected TCP connection.
`ssearch.c`	Like `ssnoauth.c`, but with certificate based authentication thrown in.
`srvrsort.c`	Shows how to use server side sorting in conjunction with the `ldap_search_ext_s()` function.
`rdentry.c`	Shows how to use `ldap_search_s()` to retrieve a particular entry from the directory. In this example, the entry "`uid=bjensen, ou=People, o=airius.com`" is retrieved and displayed.
`getattrs.c`	Just like `read.c`, but retrieves specific attributes from an entry.
`compare.c`	Shows how to use `ldap_compare_s()`, which allows you to test if a particular value is contained in an attribute of an entry.
`modattrs.c`	Shows how to use `ldap_modify_s()` to replace and add to values in an attribute.
`modrdn.c`	Shows how to use `ldap_modrdn2_s()` to change the relative distinguished name (rdn) of an entry.

Table 1-5 Synchronous example programs

Example Program	Description
`getfilt.c`	Shows how to use the `ldap_getfilter*()` family of routines, which help generate LDAP filters based on an arbitrary search string provided by a user.
`crtfilt.c`	Shows how to use the `ldap_create_filter()` function to generate LDAP filters.

Asynchronous Examples

These examples use the asynchronous LDAP calls. The general idea is that you begin an operation, and then periodically poll to see if any results have been returned.

Table 1-6 Asynchronous example programs

Example	Description
`asearch.c`	Initiates a search for entries and polls for results, printing them as they arrive.
`nsprio.c`	Like `asearch.c` but uses the PerLDAP routines to incorporate Netscape Portable Runtime (NSPR).
`add.c`	Adds an entry to the directory.
`del.c`	Deletes an entry from the directory.
`psearch.c`	Shows how to use the Persistent Search (an LDAPv3 protocol extension) to monitor a directory server for changes.
`ppolicy.c`	Attempts to bind to the directory and reports back any password expiration information received. This demonstrates how clients can process password policy information that is optionally returned by Netscape Directory Server 3.0 and later, and the iPlanet Directory Server.

Chapter 2

An Introduction to LDAP

This chapter gives a brief introduction to the Lightweight Directory Access Protocol (LDAP) and the concepts behind LDAP.

LDAP 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 into the following sections:

- How Directory Services Work
- How LDAP Servers Organize Directories
- How LDAP Clients and Servers Work
- Support for the LDAPv3 Protocol

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@airius.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@airius.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 iPlanet 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 iPlanet 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 iPlanet 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 2-1 illustrates an example of a hierarchy of entries in an LDAP directory service.

Figure 2-1 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.

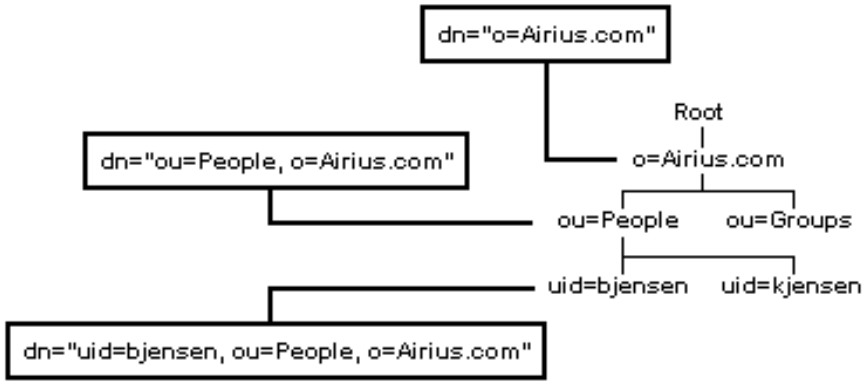
For example, this might be the distinguished name for the `bjensen` entry:

```
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 2-2 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 iPlanet 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:

- Search for and retrieve entries from the directory.
- Add new entries to the directory.
- Update entries in the directory.
- Delete entries from the directory.

- Rename 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.

Support for the LDAPv3 Protocol

Many LDAP servers support version 2 of the LDAP protocol. LDAPv2 is specified in RFC 1777. You can find a copy of this RFC at the following site:

`http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1777.txt`

The most recent proposed standard of the LDAP protocol is version 3 (LDAPv3), which is specified in RFC 2251. This RFC can be found at the following URL site:

`http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2251.txt`

While this version of the Netscape LDAP SDK for C supports LDAPv3, it “speaks” LDAPv2 by default. To override the default LDAP version, use the `ldap_set_option()` routine.

There are certain functions that are specific to LDAPv3 included in this SDK. The functions that are specific to LDAPv3 are marked as such in the Reference section of this manual.

Chapter 3

A Short Example

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

The chapter contains the following sections:

- Understanding the Sample Client
- Compiling the Sample Client
- Running the Sample Client

Understanding the Sample Client

The following is the source code for a command-line program (a console application) that retrieves the full name, last name, email address, and telephone number of Barbara Jensen.

Code Example 3-1 A console application that retrieves an entry

```
#include <stdio.h>
#include "ldap.h"

/* Adjust these setting for your own LDAP server */
#define HOSTNAME "localhost"
#define PORT_NUMBER  LDAP_PORT
#define FIND_DN "uid=bjensen, ou=People, o=airius.com"

int
main( int argc, char **argv )
{
  LDAP         *ld;
  LDAPMessage  *result, *e;
  BerElement   *ber;
```

Code Example 3-1 A console application that retrieves an entry

```
  char         *a;
  char         **vals;
  int          i, rc;

  /* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORT_NUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }

  /* Bind anonymously to the LDAP server. */
  rc = ldap_simple_bind_s( ld, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_simple_bind_s: %s\n", ldap_err2string(rc));
    return( 1 );
  }

  /* Search for the entry. */
  if ( ( rc = ldap_search_ext_s( ld, FIND_DN, LDAP_SCOPE_BASE,
    "(objectclass=*)", NULL, 0, NULL, NULL, LDAP_NO_LIMIT,
    LDAP_NO_LIMIT, &result ) ) != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_search_ext_s: %s\n", ldap_err2string(rc));
    return( 1 );
  }

  /* Since we are doing a base search, there should be only
     one matching entry.  */
  e = ldap_first_entry( ld, result );
  if ( e != NULL ) {
    printf( "\nFound %s:\n\n", FIND_DN );

    /* Iterate through each attribute in the entry. */
    for ( a = ldap_first_attribute( ld, e, &ber );
      a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {

      /* For each attribute, print the attribute name and values. */
      if ((vals = ldap_get_values( ld, e, a)) != NULL ) {
        for ( i = 0; vals[i] != NULL; i++ ) {
          printf( "%s: %s\n", a, vals[i] );
        }
        ldap_value_free( vals );
      }
      ldap_memfree( a );
    }
    if ( ber != NULL ) {
      ber_free( ber, 0 );
    }
  }
  ldap_msgfree( result );
  ldap_unbind( ld );
  return( 0 );
}
```

Compiling the Sample Client

How you compile the example program depends on the operating system on which you compile. Below are examples of compiling on Solaris and Windows NT systems.

Compiling on Solaris

The `examples` directory contains a sample UNIX makefile. You can modify the makefile to compile the example in this section by adjusting the flags specified in this file as needed. The makefile assumes that the LDAP API header files are located in the `../include` directory.

For example, you can use the following Solaris makefile for this example:

Code Example 3-2 An example Solaris makefile

```
#
# Makefile for the example in this chapter.
#
EXTRACFLAGS=
EXTRALDFLAGS=-lsocket -lnsl
LDAPLIB=ldap41 -lplc3 -lplds3 -lnspr3
INCDIR=../include
LIBDIR=../lib
LIBS=-L$(LIBDIR) $(LDAPLIB) $(EXTRALDFLAGS)
OPTFLAGS=-g
CFLAGS=$(OPTFLAGS) -I$(INCDIR) $(EXTRACFLAGS)
CC=cc
PROGS=rdentry
all:          $(PROGS)
rdentry:      rdentry.o
              $(CC) -o rdentry rdentry.o $(LIBS)
clean:
            /bin/rm -f $(PROGS) *.o a.out core
```

Compiling on Windows NT

If you are compiling this on Windows NT, set up a makefile for this application. Make sure to do the following:

- If you are using Microsoft Visual C++, create a new project workspace for a console application. Add the source file to the project.

- Set your options to include `lib` as one of the directories containing library files and `include` as one of the directories containing include files.
- Link to `nsldap32v41.lib`, the LDAP API import library for Windows.

Running the Sample Client

First, make sure your LDAP server is set up with the entry that the example attempts to find. For example, if you are running the Netscape Directory Server 4.*x*, make sure the `Arius.ldif` file is imported in the database.

Running on UNIX

If you have compiled and linked the client on a UNIX platform, make sure to set your `LD_LIBRARY_PATH` variable to the location of the `libldap41.so`, `plc3`, `plds3`, and `nspr3` library files. (As an alternative, when linking the file, specify the flag that identifies the library directories that the run-time linker should search. For example, on Solaris, use the `-R` flag to specify the location of the `libldap41.so` file.)

Running on Windows NT

If you have linked the client with the `nsldap32v41.lib` import library on Windows NT, make sure to copy the `nsldap32v41.dll` dynamic link library file to one of the following directories:

- The directory from which the application loaded.
- The current directory.
- The 32-bit Windows system directory (in Windows NT, this directory is typically `winnt\system32`).
- The Windows directory.
- The directories listed in the `PATH` environment variable.

Part 2

Writing Clients with the Netscape LDAP SDK for C

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
- Initializing an LDAP Session
- Binding and Authenticating to an LDAP Server
- Performing LDAP Operations
- Closing the Connection to the Server

The next chapter, Chapter 5, “Using the LDAP API” also includes important information on API functions.

Overview: Designing an LDAP Client

With the Netscape LDAP SDK for C, you can write a new application or enable an existing application to 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. Initialize an LDAP session. (See “Initializing an LDAP Session” for details.)

   Set any preferences that you want applied to all LDAP sessions for your client (see “Setting Preferences” for details). If you intend to use any of the LDAPv3 features (such as controls or operations), specify the LDAP version of your client, as detailed in “Specifying the LDAP Version of Your Client” (the default LDAP version used by the Netscape LDAP SDK for C is version 2).

2. If necessary, bind to the LDAP server. (See “Binding and Authenticating to an LDAP Server” for details.)
3. Perform LDAP operations, such as searching the directory or modifying entries in the directory. (See “Performing LDAP Operations” for details.)
4. When you are done, close the connection to the LDAP server. (See “Closing the Connection to the Server” for details.)

The following is a simple example of an LDAP client that requests an LDAP search operation from a server. The client connects to the LDAP server running on the local machine at port 389, searches the directory for entries with the last name “Jensen” (“sn=Jensen”), and prints out the DNs of any matching entries.

Code Example 4-1 An LDAP search operation

```
#include <stdio.h>
#include "ldap.h"

/* Specify the search criteria here. */
#define HOSTNAME "localhost"
#define PORTNUMBER 389
#define BASEDN "o=airius.com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"

int
main( int argc, char **argv )
{
  LDAP          *ld;
  LDAPMessage   *result, *e;
  char          *dn;
  int           version, rc;
  /* Print out an informational message. */
  printf( "Connecting to host %s at port %d...\n\n", HOSTNAME,
    PORTNUMBER );

  /* STEP 1: Get a handle to an LDAP connection and
    set any session preferences. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }

  /* Use the LDAP_OPT_PROTOCOL_VERSION session preference to specify
    that the client is an LDAPv3 client. */
  version = LDAP_VERSION3;
  ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

  /* STEP 2: Bind to the server.
    In this example, the client binds anonymously to the server
    (no DN or credentials are specified). */
```

Code Example 4-1 An LDAP search operation

```
  rc = ldap_simple_bind_s( ld, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_simple_bind_s: %s\n", ldap_err2string(rc));
    return( 1 );
  }

  /* Print out an informational message. */
  printf( "Searching the directory for entries\n"
    " starting from the base DN %s\n"
    " within the scope %d\n"
    " matching the search filter %s...\n\n",
    BASEDN, SCOPE, FILTER );

  /* STEP 3: Perform the LDAP operations.
    In this example, a simple search operation is performed.
    The client iterates through each of the entries returned and
    prints out the DN of each entry. */
  rc = ldap_search_ext_s( ld, BASEDN, SCOPE, FILTER, NULL, 0,
    NULL, NULL, NULL, 0, &result );
  if ( rc != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_search_ext_s: %s\n", ldap_err2string(rc));
    return( 1 );
  }
  for ( e = ldap_first_entry( ld, result ); e != NULL;
      e = ldap_next_entry( ld, e ) ) {
    if ( (dn = ldap_get_dn( ld, e )) != NULL ) {
      printf( "dn: %s\n", dn );
      ldap_memfree( dn );
    }
  }
  ldap_msgfree( result );

  /* STEP 4: Disconnect from the server. */
  ldap_unbind( ld );
  return( 0 );
}
...
```

The rest of this chapter explains how to initialize a session, authenticate your client, perform LDAP operations, and disconnect from the LDAP server.

Initializing an LDAP Session

Before you can connect to an LDAP server, you need to initialize an LDAP session. As part of this process, you create an `LDAP` structure, which contains information about the LDAP server and the LDAP session. You need to pass this `LDAP` structure (usually as a pointer) to all subsequent LDAP API functions to identify the LDAP server that you are working with.

The following sections explain how to initialize an LDAP session with a server and how to specify session preferences:

- Specifying a Single LDAP Server
- Specifying a List of LDAP Servers
- Setting Preferences

An example of initializing an LDAP session is also provided. See “Examples of Initializing an LDAP Session.”

NOTE If you plan to connect to the LDAP server over a Secure Sockets Layer (SSL), the procedure for initializing an LDAP session differs. For details, see Chapter 12, “Connecting Over SSL.”

Specifying a Single LDAP Server

To initialize the LDAP session, call the `ldap_init()` function. Pass the host name and port number of your LDAP server as arguments to this function.

If the server is using the default port for the LDAP protocol (port 389), you can pass `LDAP_PORT` as the value of the `port` parameter. For example:

```
...
LDAP *ld
...
ld = ldap_init( "directory.iplanet.com", LDAP_PORT );
```

If successful, the `ldap_init()` function returns a *connection handle*, which is a pointer to an `LDAP` structure that contains information about the connection to the LDAP server. You need to pass this pointer to the LDAP API functions for connecting, authenticating, and performing LDAP operations on a server.

For example, when you call a function to search the directory, you need to pass the connection handle as a parameter. The connection handle provides context for the connection.

Specifying a List of LDAP Servers

When initializing the LDAP session, you can specify a list of LDAP servers that you want 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.

To specify a list of LDAP servers, pass a space-delimited list of the host names as the first argument to the `ldap_init()` function. For example:

```
...
LDAP *ld
...
ld = ldap_init( "ld1.iplanet.com ld2.netscape.com
  ld3.iplanet.com", LDAP_PORT );
```

In the example above, the LDAP client will attempt to connect to the LDAP server on `ld1.iplanet.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.iplaent.com`, port 389. If that server does not respond, the client will use the server on `ld3.iplanet.com`, port 389.

If any of the servers do not use the default port specified as the second argument to `ldap_init()`, use the *host*:*port* format to specify the server name. For example:

```
...
LDAP *ld
...
ld = ldap_init( "ld1.iplanet.com ld2.iplanet.com:38900",
  LDAP_PORT );
```

In the example above, the LDAP client will attempt to connect to the LDAP server on `ld1.iplanet.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.iplanet.com`, port 38900.

Setting Preferences

To get or set the value of a preference, call the `ldap_get_option()` function or the `ldap_set_option()` function.

Both functions pass two parameters (in addition to the `ld` parameter, which represents the connection to the server):

- An `option` parameter that identifies the option that you want to get or set.
- A `value` parameter that is either a pointer to a place to put the value that you are getting or a pointer to the value that you are setting.

Note that you can set a preference for all connections created by passing NULL instead of an LDAP structure as the first argument.

For a complete list of the options and values you can get and set, see the documentation on the ldap_set_option() function.

Some of these preferences are described in the sections below:

- Setting the Restart Preference
- Specifying the LDAP Version of Your Client

Setting the Restart Preference

If communication with the LDAP server is interrupted, the result code LDAP_SERVER_DOWN is returned by your client. If you want your client to continue to attempt to communicate with the server, you can set the LDAP_OPT_RECONNECT preference for the session. If your connection is lost and this option is set, the application will attempt another bind using the same authentication to reestablish the connection.

Call the ldap_set_option() function and pass LDAP_OPT_RECONNECT as the value of the option parameter.

- If you want the client to resume LDAP I/O operations automatically, set the optdata parameter to LDAP_OPT_ON (this specifies that the same connection handle can be used to reconnect to the server).
- If you do not want the client to resume I/O operations, set the optdata parameter to LDAP_OPT_OFF (specifies that you want to create a new connection handle to connect to the server).

By default, this option is set to LDAP_OPT_OFF. Both LDAP_OPT_OFF and LDAP_OPT_ON are cast to (void *). You can pass these parameters directly to the function. For example:

```
...
ldap_set_option( ld, LDAP_OPT_RECONNECT, LDAP_OPT_ON );
...
```

Specifying the LDAP Version of Your Client

If you plan to call API functions that make use of LDAPv3 features, you should set the protocol version of your client to LDAPv3. (By default, clients built with the Netscape LDAP SDK for C identify themselves to LDAP servers as LDAPv2 clients.)

To specify the LDAP version supported by your client, call the `ldap_set_option()` function and set the `LDAP_OPT_PROTOCOL_VERSION` option to the value 3. For example:

```
...
version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
...
```

After setting this option, your client can authenticate or **bind** to the server (see "Binding and Authenticating to an LDAP Server" for details). As part of the process of binding to the server, your client sends the supported LDAP version number to the server. This allows the server to determine whether or not to enable use of LDAPv3 features.

Note that the LDAPv3 protocol allows you to perform LDAP operations without first binding to the server. An LDAPv3 server will assume that the client is LDAPv3 compliant if the client issues non-bind operations before is issues a bind.

Examples of Initializing an LDAP Session

The following section of code initializes a session with an LDAP server. The example specifies a list of LDAP servers to try: `ldap.iplanet.com:389` and `directory.iplanet.com:3890`. The example also sets a session preference that identifies the client as an LDAPv3 client.

Code Example 4-2 Initializing an LDAP session

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
int ldap_default_port, version;

/* Specify list of LDAP servers that you want to try connecting to. */
char *ldap_host = "ldap.iplanet.com directory.iplanet.com:3890";

/* If the LDAP server is running on the standard LDAP port (port 389),
 * you can use LDAP_PORT to identify the port number. */
ldap_default_port = LDAP_PORT;
...
/* Initialize the session with the LDAP servers. */
if ( ( ld = ldap_init( ldap_host, ldap_default_port ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}
```

Code Example 4-2 Initializing an LDAP session

```
/* Specify the LDAP version supported by the client. */
version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

...
/* Subsequent API calls pass ld as an argument to identify the LDAP server. */

...
```

After you initialize a session with an LDAP server, you can set session options and connect and authenticate to the LDAP server. (Note that the `ldap_init()` function does not connect to the LDAP server right away.)

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 certain access privileges.

- You are connecting to an LDAPv2 server.

  LDAPv2 servers typically require clients to bind before any operations can be performed.

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

Understanding Authentication Methods

When binding to an LDAP server, you can use a number of different methods to authenticate your client:

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

  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 *Simple Authentication and Security Layer (SASL)*, RFC 2222. For more information, refer to "Where to Find Additional Information" on page 21. Some LDAPv3 servers (including the iPlanet Directory Server 5.0) support authentication through SASL.

  For more information on using SASL mechanisms for authentication, see Chapter 13, "Using SASL Authentication."

Using Simple Authentication

If you plan to use simple authentication, call one of the following functions:

- Call the synchronous `ldap_simple_bind_s()` function if you want to wait for the bind operation to complete before the function returns. (See "Performing a Synchronous Authentication Operation.")

- Call the asynchronous `ldap_simple_bind()` function if you do not want to wait for the bind operation to complete. You can perform other work while periodically checking for the results of the bind operation. (See "Performing an Asynchronous Authentication Operation.")

Note that if you specify a DN but no password, your client will bind to the server anonymously. If you want a `NULL` password to be rejected as an incorrect password, you need to write code to perform the check before you call the `ldap_simple_bind()` or the `ldap_simple_bind_s()` functions.

If you are binding to the iPlanet Directory Server 5.0, 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."

For more information about the difference between synchronous and asynchronous functions, see "Calling Synchronous and Asynchronous Functions."

Performing a Synchronous Authentication Operation

If you want to wait for the bind operation to complete before continuing, call the `ldap_simple_bind_s()` function. This function returns `LDAP_SUCCESS` if the operation successfully completed or an LDAP result code if a problem occurred. (See the documentation for this function for a list of possible result codes returned.)

If you specify a DN but no password, your client will bind to the server anonymously. If you want a `NULL` password to be rejected as an incorrect password, you need to write code to perform the check before you call the `ldap_simple_bind_s()` function.

The following section of code uses the synchronous `ldap_simple_bind_s()` function to authenticate the user Barbara Jensen to the LDAP server. (For an example of binding anonymously, see "Binding Anonymously.")

Code Example 4-3 Synchronous authentication

```
#include <stdio.h>
#include "ldap.h"

/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "uid=bjensen,ou=People,o=airius.com"
#define BIND_PW "hifalutin"

LDAP      *ld;
int      rc;
```

Code Example 4-3 Synchronous authentication

```
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Print out an informational message. */
printf( "Binding to server %s:%d\n", HOSTNAME, PORTNUMBER );
printf( "as the DN %s ...\n", BIND_DN );

/* Bind to the LDAP server. */
rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
if ( rc != LDAP_SUCCESS ) {
  fprintf(stderr, "ldap_simple_bind_s: %s\n\n", ldap_err2string(rc));
  return( 1 );
} else {
  printf( "Bind operation successful.\n" );
}

...
/* If you want, you can perform LDAP operations here. */
...

/* Disconnect from the server when done. */
ldap_unbind( ld );
return( 0 );
...
```

Performing an Asynchronous Authentication Operation

If you want to perform other work (in parallel) while waiting for the bind operation to complete, call the `ldap_simple_bind()` function. This function sends an LDAP bind request to the server and returns a message ID identifying the bind operation.

If you specify a DN but no password, your client will bind to the server anonymously. If you want a `NULL` password to be rejected as an incorrect password, you need to write code to perform the check before you call the `ldap_simple_bind()` function.

You can check to see if your client has received the results of the bind operation from the server by calling the `ldap_result()` function and passing it the message ID.

If your client has received the results, the `ldap_result()` function passes back the results of the bind operation in an `LDAPMessage` structure. To get error information from this structure, you can pass it to the `ldap_parse_result()` function.

The `ldap_parse_result()` function gets the LDAP result code of the operation and any error messages sent back from the server. This function also retrieves any controls sent back from the server.

(The iPlanet Directory Server 5.0 may return a control if the user's password has expired or will expire in the near future. For more information, see "Using Password Policy Controls.")

The following section of code uses the asynchronous `ldap_simple_bind()` function to authenticate the user Barbara Jensen to the LDAP server. (For an example of binding anonymously, see "Binding Anonymously.")

Code Example 4-4 Asynchronous authentication

```
#include <stdio.h>
#include "ldap.h"

void do_other_work();
int global_counter = 0;
...

#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "uid=bjensen,ou=People,o=airius.com"
#define BIND_PW "hifalutin"

...
LDAP            *ld;
LDAPMessage     *res;
int             msgid = 0, rc = 0, parse_rc = 0, finished = 0;
char            *matched_msg = NULL, *error_msg = NULL;
char            **referrals;
LDAPControl     **serverctrls;
struct timeval  zerotime;

/* Specify the timeout period for ldap_result(),
   which specifies how long the function should block when waiting
   for results from the server. */
zerotime.tv_sec = zerotime.tv_usec = 0L;

/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Print out an informational message. */
printf( "Binding to server %s:%d\n", HOSTNAME, PORTNUMBER );
printf( "as the DN %s ...\n", BIND_DN );

/* Send an LDAP bind request to the server. */
msgid = ldap_simple_bind( ld, BIND_DN, BIND_PW );
```

Code Example 4-4 Asynchronous authentication

```
/* If the returned message ID is less than zero, an error occurred. */
if ( msgid < 0  ) {
  rc = ldap_get_lderrno( ld, NULL, NULL );
  fprintf(stderr, "ldap_simple_bind : %s\n", ldap_err2string(rc));
  ldap_unbind( ld );
  return( 1 );
}

/* Check to see if the bind operation completed. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {
  /* If ldap_result() returns -1, error occurred. */
  case -1:
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return ( 1 );

    /* If ldap_result() returns 0, the timeout (specified by the
      timeout argument) has been exceeded before the client received
      the results from the server. Continue calling ldap_result()
      to poll for results from the server. */
  case 0:
    break;

  default:
    /* The client has received the result of the bind operation. */
    finished = 1;

    /* Parse this result to determine if the operation was successful.
      Note that a non-zero value is passed as the last parameter,
      which indicates that the LDAPMessage structure res should be
      freed when done.  (No need to call ldap_msgfree().) */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }
    /* Check the results of the operation. */
    if ( rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_simple_bind: %s\n",
        ldap_err2string( rc ) );

      /* If the server sent an additional error message,
        print it out. */
      if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
```

Code Example 4-4 Asynchronous authentication

```
      /* If an entry specified by a DN could not be found,
        the server may also return the portion of the DN
        that identifies an existing entry.
        (See “Receiving the Portion of the DN Matching an Entry”
        for an explanation.) */
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
      }
      ldap_unbind( ld );
      return( 1 );
    } else {
      printf( "Bind operation successful.\n" );
      printf( "Counted to %d while waiting for bind op.\n",
        global_counter );
    }
    break;
  }
  /* Do other work here while waiting for results from the server. */
  if ( !finished ) {
    do_other_work();
  }
}

...
/* If you want, you can perform LDAP operations here. */
...

/* Disconnect from the server when done. */
ldap_unbind( ld );
return( 0 );
...
/* Function that does work while waiting for results from the server. */
void do_other_work() {
  global_counter++;
}
...
```

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.

With LDAPv2, the client is required to send a bind request, even when binding anonymously (binding without specifying a name or password). With LDAPv3, the client is no longer required to bind to the server if the client does not need to authenticate.

To bind as an anonymous user, call `ldap_simple_bind()` or `ldap_simple_bind_s()`, and pass `NULL` values for the `who` and `passwd` parameters. For example:

```
...
rc = ldap_simple_bind_s( ld, NULL, NULL );
...
```

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 the server's access control allows these operations).

To perform LDAP operations, call these API functions:

- To search for entries in the directory, call `ldap_search_ext()` or `ldap_search_ext_s()`. (See Chapter 6, "Searching the Directory" for details.)
- To determine whether or not an attribute contains a specified value, call `ldap_compare_ext()` or `ldap_compare_ext_s()`. (See "Comparing the Value of an Attribute" for details.)
- To add entries to the directory, call `ldap_add_ext()` or `ldap_add_ext_s()`. (See "Adding a New Entry" for details.)
- To modify entries in the directory, call `ldap_modify_ext()` or `ldap_modify_ext_s()`. (See "Modifying an Entry" for details.)
- To delete entries from the directory, call `ldap_delete_ext()` or `ldap_delete_ext_s()`. (See "Deleting an Entry" for details.)
- To change the DNs of entries in the directory, call `ldap_rename()` or `ldap_rename_s()`. (See "Changing the DN of an Entry" for details.)

Most LDAP operations can be performed synchronously or asynchronously. The functions with names ending in `_s` are synchronous functions, and the remaining functions are asynchronous functions. (For more information on the distinction between calling synchronous and asynchronous functions, see "Calling Synchronous and Asynchronous Functions.")

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.

To close a connection to an LDAP server, call one of the following functions:

- `ldap_unbind()`
- `ldap_unbind_s()`
- `ldap_unbind_ext()`

Both `ldap_unbind()` and `ldap_unbind_s()` are synchronous functions. These functions are identical; they use different names so that each authentication function (`ldap_simple_bind()` and `ldap_simple_bind_s()`) has a corresponding function for closing the server connection.

After you close the connection, you can no longer use the LDAP structure. Calling any of the unbind functions frees the `LDAP` structure from memory.

The following code closes the current connection with the LDAP server:

Code Example 4-5 Closing an LDAP server connection

```
#include <stdio.h>
#include "ldap.h"
...
LDAP      *ld;
int      rc;
...
/* After completing your LDAP operations with the server, close
  the connection. */
rc = ldap_unbind( ld );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_unbind: %s\n", ldap_err2string( rc ) );
}
...
```

The `ldap_unbind_ext()` function allows you to explicitly include both server and client controls in your unbind request. However, note that since there is no server response to an unbind request, there is no way to receive a response from a server control that is included with your unbind request.

Chapter 5

Using the LDAP API

This chapter covers some of the general LDAP API functions that are commonly used when writing LDAP clients. This chapter includes instructions on getting version information, freeing memory, checking for errors, and requesting synchronous and asynchronous functions.

- Getting Information About the SDK
- Managing Memory
- Reporting Errors
- Calling Synchronous and Asynchronous Functions
- Handling Referrals
- Setting Up an In-Memory Cache
- Handling Failover

Getting Information About the SDK

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

To get this version information, call the `ldap_get_option()` function using the following option:

```
ldap_get_option (..., LDAP_OPT_API_INFO, ...) ;
```

The following example illustrates retrieving the version information:

Code Example 5-1 Retrieving LDAP version information

```
#include <stdio.h>
#include “ldap.h”

main()
{

LDAPAPIInfo         ldapi;
LDAPAPIFeatureInfo  fi;
int                 i;
int                 rc;
LDAP                *ld;

memset( &ldapi, 0, sizeof(ldapi));
ldapi.ldapai_info_version = LDAP_API_INFO_VERSION;

if ((rc = ldap_get_option( ld, LDAP_OPT_API_INFO, &ldapi)) != 0) {
        printf(“Error: ldap_get_option (rc: %d)\n”, rc);
        exit(0);
}

printf(“LDAP Library Information -\n”
        “    Highest supported protocol version: %d\n”
        “    LDAP API revision:                  %d\n”
        “    API vendor name:                    %s\n”
        “    Vendor-specific version:            %.2f\n”,
        ldapi.ldapai_protocol_version, ldapi.ldapai_api_version,
        ldapi.ldapai_vendor_name,
        (float)ldapi.ldapai_vendor_version / 100.0 );

if ( ldapi.ldapai_extensions != NULL ) {
        printf(“    LDAP API Extensions:\n”);

        for ( i = 0; ldapi.ldapai_extensions[i] != NULL; i++ )  {
            printf(“        %s”, ldapi.ldapai_extensions[i] );
            fi.ldapaif_info_version = LDAP_FEATURE_INFO_VERSION;
            fi.ldapaif_name = ldapi.ldapai_extensions[i];
            fi.ldapaif_version = 0;

            if ( ldap_get_option( NULL, LDAP_OPT_API_FEATURE_INFO, &fi )
                    != 0 ) {
                printf(“Error: ldap_get_option( NULL,”
                       “ LDAP_OPT_API_FEATURE_INFO, ... ) for %s failed”
                       “ (Feature Info version: %d)\n”,
                       fi.ldapaif_name, fi.ldapaif_info_version );
            } else {
                printf(“ (revision %d)\n”, fi.ldapaif_version);
            }
        }
    }
   printf(“\n”);
}
```

Managing Memory

Several of the LDAP API functions allocate memory when called. When you have finished working with data allocated by these functions, you should free the memory.

Table 5-1 lists some of the API functions that allocate memory and the corresponding functions that you must call to free the memory when you are done.

Table 5-1 API functions that allocate and free memory

Functions to free memory	Type of memory freed
`ldap_unbind()` or `ldap_unbind_s()`	Frees `LDAP` structures allocated by calling `ldap_init()`.
`ldap_msgfree()`	Frees `LDAPMessage` structures allocated by calling `ldap_result()` or `ldap_search_ext_s()`.
`ldap_ber_free()`	Frees `BerElement` structures allocated by calling `ldap_first_attribute()`.
`ldap_value_free()`	Frees `char **` arrays structures allocated by calling `ldap_get_values()`.
`ldap_value_free_len()`	Frees Arrays of `berval` structures allocated by calling `ldap_get_values_len()`.
`ber_bvfree()`	Frees `berval` structures allocated by calling `ldap_extended_operation_s()`, `ldap_parse_extended_result()`, `ldap_parse_sasl_bind_result()`, and `ldap_sasl_bind_s()`.
`ldap_free_friendlymap()`	Frees `FriendlyMap` structures allocated by calling `ldap_friendly_name()`.
`ldap_free_urldesc()`	Frees `LDAPURLDesc` structures allocated by calling `ldap_url_parse()`.
`ldap_getfilter_free()`	Frees `LDAPFiltDesc` structures allocated by calling `ldap_init_getfilter()` or `ldap_init_getfilter_buf()`.
`ldap_mods_free()`	Frees `LDAPMod **` arrays and structures allocated by functions that you call when you add or modify entries.
`ldap_free_sort_keylist()`	Frees `LDAPsortkey **` arrays structures allocated by calling `ldap_create_sort_keylist()`.

Table 5-1 API functions that allocate and free memory

Functions to free memory	Type of memory freed
`ldap_control_free()`	Frees `LDAPControl` structures allocated by calling `ldap_create_sort_control()` or `ldap_create_persistentsearch_control()`.
`ldap_controls_free()`	Frees `LDAPControl **` arrays structures allocated by calling `ldap_get_entry_controls()`, `ldap_parse_result()`, or `ldap_parse_reference()`.
`ldap_memfree()`	Any other types of memory that you allocate (this function is a general function for freeing memory).

See the descriptions of individual functions in Chapter 18, “Function Reference” for more information about memory management.

Reporting Errors

In the LDAP protocol, the success or failure of an operation is specified by an LDAP result code sent back to the client. A result code of `0` normally indicates that the operation was successful whereas a non-zero result code usually indicates that an error occurred.

The following sections explain more about handling and reporting errors:

- Getting Information About the Error
- Getting the Error Message
- Setting Error Codes
- Printing Out Error Messages

Getting Information About the Error

When an error occurs in an LDAP operation, the server sends the following information back to the client:

- 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 server may also return the portion of the DN that identifies an existing entry. (See "Receiving the Portion of the DN Matching an Entry" for an explanation.)

There are two ways you can get this information back from the server:

- If you are calling asynchronous functions, you can get the information from the `LDAPMessage` structure representing the result returned from the server.

  For details, see "Getting the Information from an LDAPMessage Structure."

- In situations where you do not have an `LDAPMessage` structure (for example, if you are calling functions that do not interact with the server), you can get error information from the connection handle. For details, see "Getting the Information from an LDAP Structure."

Receiving the Portion of the DN Matching an Entry

According to the LDAPv3 protocol, if a server returns an `LDAP_NO_SUCH_OBJECT`, `LDAP_ALIAS_PROBLEM`, `LDAP_INVALID_DN_SYNTAX`, or `LDAP_ALIAS_DEREF_PROBLEM` result code, the LDAP server should also send back the portion of DN that matches the entry that is closest to the requested entry.

For example, suppose the LDAP server processes a request to modify the entry with the DN "`uid=bjensen,ou=Contractors,o=airius.com`" but that entry does not exist in the directory.

- If the entry with the DN "`ou=Contractors,o=airius.com`" does exist, the server sends this portion of the DN ("`ou=Contractors,o=airius.com`") with the result code `LDAP_NO_SUCH_OBJECT`.

- If the entry with the DN "`ou=Contractors,o=airius.com`" does exist either, but the entry with the DN "`o=airius.com`" does exist, the server sends "`o=airius.com`" back to the client with the result code `LDAP_NO_SUCH_OBJECT`.

  Basically, the server moves back up the directory tree (one DN component at a time) until it can find a DN that identifies an existing entry.

Getting the Information from an LDAPMessage Structure

If you have requested the operation through an asynchronous function, not a synchronous function (see "Calling Synchronous and Asynchronous Functions" for the difference between these functions), you can get the result of the operation from the server by calling the `ldap_result()` function.

This function passes the result as an LDAPMessage structure. You can get information from this structure by calling the ldap_parse_result() function:

```
LDAP_API(int) LDAP_CALL ldap_parse_result( LDAP *ld,
  LDAPMessage *res, int *errcodep, char **matcheddnp,
  char **errmsgp, char ***referralsp,
  LDAPControl ***serverctrlsp, int freeit );
```

The different types of information are returned in the following parameters of this function:

- The LDAP result code is the errcodep argument.
- Additional information from the server is passed back as the errmsgp argument.
- In cases when the server cannot find an entry from a DN, the portion of the DN that identifies an existing entry is passed back as the matcheddnp argument. (See "Receiving the Portion of the DN Matching an Entry" for details.)

Note that you can also get the error message describing the LDAP result code by using the ldap_err2string() function. (See the section "Getting the Error Message" for details.)

For a listing and descriptions of the different LDAP result codes, see Chapter 19, "Result Codes."

The following section of code gets and prints information about an error returned from the server.

Code Example 5-2 Receiving and printing error codes from an LDAPMessage structure

```
#include <stdio.h>
#include "ldap.h"
...
LDAP              *ld;
LDAPMessage       *res;
int               msgid = 0, rc = 0, parse_rc = 0, finished = 0;
char              *matched_msg = NULL, *error_msg = NULL;
char              **referrals;
LDAPControl       **serverctrls;
struct timeval    zerotime;
...
while ( !finished ) {
  /* Check to see if the server returned a result. */
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {
  ...
  default:
    /* The client has received the result of the LDAP operation. */
```

Code Example 5-2 Receiving and printing error codes from an LDAPMessage structure

```
    finished = 1;

    /* Parse this result to determine if the operation was successful.
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );

    /* Verify that the result was parsed correctly. */
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result error: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }

    /* Check the results of the operation. */
    if ( rc != LDAP_SUCCESS ) {

      /* Print the error message corresponding to the result code. */
      fprintf( stderr, "Error: %s\n",
        ldap_err2string( rc ) );

      /* If the server sent an additional message, print it out. */
      if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }

      /* If the server cannot find an entry with the specified DN,
        it may send back the portion of the DN that matches
        an existing entry, For details, see
        "Receiving the Portion of the DN Matching an Entry". */
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
      }

      /* Disconnect and return. */
      ldap_unbind( ld );
      return( 1 );
    }
...
```

Getting the Information from an LDAP Structure

In situations where you don't get an `LDAPMessage` structure (for example, if you are calling functions that do not interact with the server), you can get error information from the connection handle (the `LDAP` structure).

To get information about the last error that has occurred, call the `ldap_get_lderrno()` function:

```
LDAP_API(int) LDAP_CALL ldap_get_lderrno(LDAP *ld,
  char **m, char **s);
```

The different types of information are returned in the following ways:

- The LDAP result code is returned by this function.
- Additional information from the server is passed back as the `s` argument.
- In cases when the server cannot find an entry from a DN, the portion of the DN that identifies an existing entry is passed back as the `m` argument. (See "Receiving the Portion of the DN Matching an Entry" for details.)

If you do not need to use the parameters returned by the `ldap_get_lderrno()` function, set the parameters to a `NULL` value. For example:

```
ldap_get_lderrno( ld, NULL, NULL );
```

The following section of code gets and prints information about an error.

Code Example 5-3 Getting an error message from an LDAP structure

```
#include <stdio.h>
#include "ldap.h"
...
LDAP     *ld;
char*    *error_msg = NULL, *matched_msg = NULL;
int      rc;
...
rc = ldap_get_lderrno( ld, &matched_msg, &error_msg );
fprintf( stderr, "ldap_result error: %s\n", ldap_err2string( rc ) );
if ( error_msg != NULL && *error_msg != '\0' ) {
  fprintf( stderr, "%s\n", error_msg );
}

/* If the server cannot find an entry with the specified DN,
  it may send back the portion of the DN that matches
  an existing entry, For details, see
  "Receiving the Portion of the DN Matching an Entry". */
if ( matched_msg != NULL && *matched_msg != '\0' ) {
  fprintf( stderr,
    "Part of the DN that matches an existing entry: %s\n",
    matched_msg );
}
...
```

Getting the Error Message

If you have an error code and want to retrieve its corresponding error message, call the `ldap_err2string()` function. The function returns a pointer to the error message. For example:

Code Example 5-4 Returning an error message from an error code

```
#include <stdio.h>
#include "ldap.h"
...
int      rc;
...
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "Error: %s\n", ldap_err2string( rc ) );
}
...
```

Note that the pointer returned by this function is a pointer to static data; do not free this string.

Setting Error Codes

When an LDAP operation is performed, the error information from the operation is specified in the `LDAP` structure. If you want to set error codes and error information in the `LDAP` structure, call the `ldap_set_lderrno()` function.

The following section of code sets the `LDAP_PARAM_ERROR` error code in the `LDAP` structure.

Code Example 5-5 Setting an error code

```
#include "ldap.h"
...
LDAP *ld;
char *errmsg = "Invalid parameter";
...
if ( ldap_my_function() != LDAP_SUCCESS ) {
  ldap_set_lderrno( ld, LDAP_PARAM_ERROR, NULL, errmsg );
  return( 1 );
}
...
```

Printing Out Error Messages

To print out the error message describing the last error that occurred, call the `ldap_get_lderrno()` function.

This example prints a message if a function fails to delete an entry in the server.

Code Example 5-6 Printing error codes

```
#include "ldap.h"
...
int lderr;
char * errmsg;
LDAP *ld;
char *dn = "uid=bjensen, ou=People, o=airius.com";
...
if ( ldap_delete_s( ld, dn ) != LDAP_SUCCESS ) {
  lderr = ldap_get_lderrno (ld, NULL, &errmsg);
  if ( errmsg, != NULL ) {
    fprintf(stderr, "ldap_delete_s: %s\n", errmsg );
    ldap_memfree( errmsg );
  }
  return( 1 );
}
...
```

In the preceding example, the client prints out this error message if it does not have access permissions to delete the entry:

```
ldap_delete_s: Insufficient access
```

Calling Synchronous and Asynchronous Functions

You can perform the operation as a synchronous or asynchronous operation. For example, to search the directory, you can call either the synchronous function `ldap_search_ext_s()`, or the asynchronous function `ldap_search_ext()`. In general, the synchronous functions have names ending with the characters `_s` (for example, `ldap_search_ext_s()`).

- For details on calling synchronous functions, see “Calling Synchronous Functions.”

- For details on calling asynchronous functions, see “Calling Asynchronous Functions.”

Calling Synchronous Functions

When you call a *synchronous function*, your client waits for the operation to complete before executing any subsequent lines of code. Synchronous functions return LDAP_SUCCESS if successful and an LDAP error code if not successful.

This example calls a synchronous function to delete an entry in the directory.

Code Example 5-7 Calling synchronous functions

```
#include <stdio.h>
#include "ldap.h"
...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform an LDAP delete operation. */
rc = ldap_delete_ext_s( ld, DELETE_DN, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_delete_ext_s: %s\n", ldap_err2string( rc ) );
  ldap_get_lderrno( ld, &matched_msg, &error_msg );
  if ( error_msg != NULL && *error_msg != ’\0’ ) {
    fprintf( stderr, "%s\n", error_msg );
  }

  /* If the server cannot find an entry with the specified DN,
    it may send back the portion of the DN that matches
    an existing entry, For details, see
    “Receiving the Portion of the DN Matching an Entry” */
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
} else {
  printf( "%s deleted successfully.\n", DELETE_DN );
}
...
```

To see additional sample programs that call synchronous functions, see the source files in the examples directory of the Netscape LDAP SDK for C.

Calling Asynchronous Functions

When you call an **asynchronous function**, your client does not need to wait for the operation to complete. The client can continue performing other tasks (such as initiating other LDAP operations) while the LDAP operation is executing.

An asynchronous function passes back a unique message ID that identifies the operation being performed. You can pass this message ID to the `ldap_result()` function to check the status of the LDAP operation.

This section explains how to call an asynchronous function and check the results of the LDAP operation. The following topics are covered:

- Checking if the LDAP Request was Sent
- Getting the Server Response
- Getting Information from the Server Response
- Canceling an Operation in Progress

The section also includes an example of calling an asynchronous function (see “Example of Calling an Asynchronous Function”).

To see sample programs that call asynchronous functions see the source files in the `examples` directory of the Netscape LDAP SDK for C.

Checking if the LDAP Request was Sent

Asynchronous functions return an LDAP result code indicating whether or not the LDAP request was successfully sent to the server. If the function returns `LDAP_SUCCESS`, the function successfully sent the request to the server.

For example, the following section of code send an LDAP delete request to the server. The example checks if the result was successfully sent.

Code Example 5-8 Asynchronous return codes

```
#include <stdio.h>
#include "ldap.h"
...
/* Change these as needed. */
#define DELETE_DN "uid=wjensen,ou=People,o=airius.com"
...
LDAP       *ld;
int        rc, msgid;
...
/* Send an LDAP delete request to the server. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
```

Code Example 5-8 Asynchronous return codes

```
if ( rc != LDAP_SUCCESS ) {
  /* If the request was not sent successfully,
    print an error message and return. */
  fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}
...
```

Getting the Server Response

If the request was successfully sent, the function passes back the message ID of the LDAP operation. You can use this message ID to determine if the server has sent back the results for this specific operation.

To check for results, call the `ldap_result()` function, and pass the message ID as a parameter. You can also specify a time-out period to wait for results from the server.

The function returns one of the following values:

- `-1` indicates that an error occurred.
- `0` indicates that the time-out period has been exceeded and the server has not yet sent a response back to your client.
- Any other value indicates that the server has sent a response for the requested operation back to your client.

You can set up a loop to poll for results while doing other work. For example, suppose you defined a function that did work while waiting for the LDAP operation to complete and the server to send a response back to your client:

```
int global_counter = 0;
void
do_other_work()
{
  global_counter++;
}
```

You can set up a `while` loop to call your function when you are not checking for the server's response:

Code Example 5-9 Getting response from a server

```
#include <stdio.h>
#include "ldap.h"
...
LDAP              *ld;
LDAPMessage       *res;
LDAPControl       **serverctrls;
char              *matched_msg = NULL, *error_msg = NULL;
char              **referrals;
int               rc, parse_rc, msgid, finished = 0;
struct timeval    zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send an LDAP delete request to the server. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
...
/* Poll the server for the results of the LDAP operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );

  /* Check to see if a result was received. */
  switch ( rc ) {
  case -1:
    .../* An error occurred. */...
  case 0:
    /* The timeout period specified by zerotime was exceeded.
      This means that the server has still not yet sent the
      results of the delete operation back to your client.
      Break out of this switch statement, and continue calling
      ldap_result() to poll for results. */
  default:
    finished = 1;
    .../* Your client received a response from the server. */...
  }

  /* Do other work while waiting. This is
    called if ldap_result() returns 0 (before you continue
    to the top of the loop and call ldap_result() again). */
  if ( !finished ) {
    do_other_work();
  }
  ...
}
...
```

Getting Information from the Server Response

If the `ldap_result()` function gets the response sent from the server, the `result` parameter passes back a pointer to an `LDAPMessage` structure. This structure contains the server's response, which can include the following information:

- An LDAP result code specifying the result of the operation you requested.
- An additional error message (optional) sent back from the server.
- If the server was not able to find the entry specified by a DN, the portion of the DN that identifies an existing entry (see "Receiving the Portion of the DN Matching an Entry" for details).
- A set of referrals, if the server's directory does not contain the requested entries and if the server is configured to refer clients to other servers.
- A set of server response controls applicable to the operation you requested (see Chapter 14, "Working with LDAP Controls" for information on LDAPv3 controls).

Note that when processing LDAP search operations, the server can also send back individual entries matching the search, individual search references, and chains of entries and search references. For information on processing these types of results from the server, see Chapter 6, "Searching the Directory."

To get information from a server response, call the `ldap_parse_result()` function:

```
LDAP_API(int) LDAP_CALL
ldap_parse_result( LDAP *ld, LDAPMessage *res, int *errcodep,
  char **matcheddnp, char **errmsgp, char ***referralsp,
  LDAPControl ***serverctrlsp, int freeit );
```

You can get the following information from parameters of this function:

- `errcodep` is the LDAP result code of the operation that the server finished processing.
- `matcheddnp` is the portion of the DN that matches an existing entry, if the server is not able to find an entry for a DN that you've specified (for details, see "Receiving the Portion of the DN Matching an Entry").
- `errmsgp` is an additional error message that the server can send to your client.
- `referralsp` is a set of referrals sent back to your client by the server, if you've requested an entry that is not part of the directory tree managed by the server and if the server is configured to refer clients to other LDAP servers.

- `serverctrlsp` is a set of server response controls applicable to the LDAP operation.

When you are done, you should call `ldap_msgfree()` to free the `LDAPMessage` structure unless the structure is part of a chain of results. If you pass a non-zero value for the `freeit` parameter, the structure is automatically freed after the information is retrieved.

The result code returned by this function is not the same as the result code of the operation (`errcodep`). The result code returned by this operation indicates the success or failure of parsing the `LDAPMessage` structure.

For example, the following section of code retrieves error information from an `LDAPMessage` structure returned by the `ldap_result()`.

Code Example 5-10 Regiving error information from an LDAPMessage structure

```
#include <stdio.h>
#include "ldap.h"
...
LDAP              *ld;
LDAPMessage       *res;
LDAPControl       **serverctrls;
char              *matched_msg = NULL, *error_msg = NULL;
char              **referrals;
int               rc, parse_rc, msgid, finished = 0;
struct timeval    zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;
...
rc = ldap_result( ld, msgid, 0, &zerotime, &res );

/* Check to see if a result was received. */
switch ( rc ) {
case -1:
  ...
case 0:
  ...
default:
  ...
  /* Call ldap_parse_result() to get information from the results
    received from the server. */
  parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
    &error_msg, &referrals, &serverctrls, 1 );

  /* Make sure the results were parsed successfully. */
  if ( parse_rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_parse_result: %s\n",
      ldap_err2string( parse_rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
```

Code Example 5-10 Regiving error information from an LDAPMessage structure

```
  /* Check the results of the LDAP operation. */
  if ( rc != LDAP_SUCCESS ) {
    fprintf(stderr, "Error: %s\n", ldap_err2string(rc));
    if ( error_msg != NULL & *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    /* If the server returned the portion of the DN
      that identifies an existing entry,
      print it out. (For details, see
      "Receiving the Portion of the DN Matching an Entry") */
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
  } else {
    printf( "Operation completed successfully" );
  }
}
...
```

Canceling an Operation in Progress

If you need to cancel the LDAP operation, call the `ldap_abandon_ext()` function. The function returns `LDAP_SUCCESS` if successful or an LDAP result code if an error occurs.

Once you cancel an LDAP operation, you cannot retrieve the results of that operation. (In other words, calling `ldap_result()` does not return any results.)

Example of Calling an Asynchronous Function

The following section of code calls an asynchronous function to delete an entry in the directory. The code calls `ldap_result()` within a loop to poll the results of the LDAP delete operation.

Code Example 5-11 Deleting an entry using an asynchronous function call

```
#include <stdio.h>
#include "ldap.h"
...
void do_other_work();
int global_counter = 0;
...
/* Change these as needed. */
```

Code Example 5-11 Deleting an entry using an asynchronous function call

```
#define DELETE_DN "uid=wjensen,ou=People,o=airius.com"
...
LDAP              *ld;
LDAPMessage       *res;
LDAPControl       **serverctrls;
char              *matched_msg = NULL, *error_msg = NULL;
char              **referrals;
int               rc, parse_rc, msgid, finished = 0;
struct timeval    zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send an LDAP delete request to the server. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}

/* Poll the server for the results of the delete operation. */
while ( !finished ) {
  /* Call ldap_result() to get the results of the delete operation.
    ldap_result() blocks for the time period
    specified by the timeout argument (set to
    zerotime here) while waiting for the result
    from the server. */
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );

  /* Check to see if a result was received. */
  switch ( rc ) {
  case -1:
    /* If ldap_result() returned -1, an error occurred. */
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );

  case 0:
    /* The timeout period specified by zerotime was exceeded.
      This means that the server has still not yet sent the
      results of the delete operation back to your client.
      Break out of this switch statement, and continue calling
      ldap_result() to poll for results. */
    break;

  default:
    /* ldap_result() got the results of the delete operation
      from the server. No need to keep polling. */
    finished = 1;

    /* Call ldap_parse_result() to get information from the results
      received from the server. Note the last
      argument is a non-zero value. This means after the
```

Code Example 5-11 Deleting an entry using an asynchronous function call

```
      function retrieves information from the
      LDAPMessage structure , the structure is freed.
      (You don't need to call ldap_msgfree() to free the structure.)
    */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }

    /* Check the results of the LDAP delete operation. */
    if ( rc != LDAP_SUCCESS ) {
      fprintf(stderr, "ldap_delete_ext: %s\n", ldap_err2string(rc));
      if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
      /* Print the portion of a specified DN
        that matches an existing entry, if
        returned by the server. (For details, see
        "Receiving the Portion of the DN Matching an Entry.") */
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
      }
    } else {
      printf( "%s deleted successfully.\n"
        "Counted to %d while waiting for the delete operation.\n",
        DELETE_DN, global_counter );
    }
  }

  /* Do other work while waiting for the results of the
    delete operation. */
  if ( !finished ) {
    do_other_work();
  }
}
ldap_unbind( ld );
return 0;
...

/* Perform other work while polling for results. */
void
do_other_work()
{
  global_counter++;
}
...
```

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, the server sends back an `LDAP_NO_SUCH_OBJECT` result code.
- If the server is configured to refer you to another LDAP server, the server sends a referral back to your client. This consists of the result code (`LDAP_PARTIAL_RESULTS` for LDAPv2 clients, `LDAP_REFERRAL` for LDAPv3 clients) and one or more LDAP URLs.

  For LDAPv2 clients, the URLs are included in the error message that the server sends to the client. For LDAPv3 clients, the URLs are included in a separate section of the result.

  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 the operation from that server. (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, your client returns the result code sent from the server (`LDAP_PARTIAL_RESULTS` or `LDAP_REFERRAL`). You can get the LDAP URLs from the result by calling the `ldap_parse_result()` function.

By default, clients built with the Netscape LDAP SDK for C 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 your client retrieves search references from the server, one of the following occurs:

- If your client handles referrals automatically, the LDAP API library retrieves each search reference, binds to the server identified in the reference (see "Binding When Following Referrals" for information on specifying the DN and password for binding), and retrieves the entry.
- If your client does not handle referrals automatically, the LDAP API library simply adds the search reference to the chain of search results. The search reference is a message of the type `LDAP_RES_SEARCH_REFERENCE`.

  You can get the search references from a chain of results by calling the `ldap_first_reference()` and `ldap_next_reference()` functions. You can also call the `ldap_first_message()` and `ldap_next_message()` functions to get each message in the search results, then call the `ldap_msgtype()` function to determine if the message is of the type `LDAP_RES_SEARCH_REFERENCE`.

Enabling or Disabling Referral Handling

By default, Netscape LDAP SDK for C clients automatically follow these referrals to other servers.

To change the way referrals are handled, call the `ldap_set_option()` function and pass `LDAP_OPT_REFERRALS` as the value of the `option` parameter.

- To prevent the client from automatically following referrals, set the `optdata` parameter to `LDAP_OPT_OFF`.
- If you want the client to automatically follow referrals again, set the `optdata` parameter to `LDAP_OPT_ON`.

Note that both `LDAP_OPT_OFF` and `LDAP_OPT_ON` are cast to `(void *)`. You can pass these parameters directly to the function (see the example below).

The following example prevents the client from automatically following referrals to other LDAP servers.

Code Example 5-12 Disabling referrals

```
#include <stdio.h>
#include "ldap.h"
...
LDAP    *ld;
int     rc;
char    *host = "localhost";
...
/* Initialize a session with the LDAP server ldap.netscape.com:389. */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Never follow referrals. */
if (ldap_set_option(ld,LDAP_OPT_REFERRALS,LDAP_OPT_OFF)!=LDAP_SUCCESS){
  rc = ldap_get_lderrno( ld, NULL, NULL );
  fprintf( stderr, "ldap_set_option: %s\n",
    ldap_err2string( rc );
  return( 1 );
}
...
```

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, the LDAP result code `LDAP_REFERRAL_LIMIT_EXCEEDED` is returned.

To set this limit, pass `LDAP_OPT_REFERRAL_HOP_LIMIT` as the value of the `option` parameter and pass the maximum number of hops as value of the `optdata` parameter.

By default, the maximum number of hops is 5.

Binding When Following Referrals

If the session 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 user name or password specified) when following referrals.

This section explains how to set up your client to authenticate with a selected DN and credentials when following referrals. Topics include:

- How Binding Works When Following Referrals
- Defining the Rebind Function
- Registering the Rebind Function

How Binding Works 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 rebind function of the type `LDAP_REBINDPROC_CALLBACK`. Then, you specify that your function should be used if binding to other servers when following referrals.

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 calls the rebind function (the function specified by the `LDAP_OPT_REBIND_FN` option), passing `0` as the `freeit` argument.
3. The rebind function sets the `dnp`, `passwdp`, and `authmethodp` arguments to point to the following information:
    - The `dnp` argument is set to point to the DN to be used to authenticate to the new LDAP server.
    - The `passwdp` argument is set to point to the credentials for this DN.

- The `authmethodp` argument is set to point to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).

4. If successful, the rebind function returns `LDAP_SUCCESS`, and referral processing continues.

   (If any other value is returned, referral processing stops, and that value is returned as the result code for the original LDAP request.)

5. The client gets the DN, credentials, and authentication method from the arguments of the rebind function and uses this information to authenticate to the new LDAP server.

6. The client calls the rebind function again, passing 1 as the `freeit` argument.

7. The rebind function frees any memory allocated earlier to specify the DN and credentials.

Defining the Rebind Function

You need to define a rebind function that does the following:

- If `freeit` is 0, set the following pointers:
  - Set `dnp` to point to the DN to be used for authentication.
  - Set `passwdp` to point to the credentials to be used for authentication.
  - Set `authmethodp` to point to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).

  You can make use of the `arg` argument, which is a pointer to the argument specified in the `ldap_set_rebind_proc()` function.

  If successful, return `LDAP_SUCCESS`; otherwise, return the appropriate LDAP error code.

- If `freeit` is `1`, free any memory that you allocated to create the DN and credentials.

You need to write a function that has the following prototype:

```
int LDAP_CALL LDAP_CALLBACK rebindproc( LDAP *ld, char **dnp,
  char **passwdp, int *authmethodp, int freeit, void *arg );
```

The parameters for this prototype are described below:

Table 5-2 LDAP_CALL_LDAP_CALLBACK parameters

Parameter Name	Description
`ld`	The connection handle to the LDAP server.
`dnp`	A pointer to the distinguished name of the user (or entity) who wants to perform the LDAP operations. Your function needs to set this value.
`passwdp`	A pointer to the user's (or entity's) password. Your function needs to set this value.
`authmethodp`	A pointer to the method of authentication. Your function needs to set this value.
`freeit`	Specifies whether or not to free the memory allocated by the previous `rebindproc()` function call (in the event that this function is called more than once). If `freeit` is set to a non-zero value, your function should free the memory allocated by the previous call.
`arg`	A pointer to data that can be passed to your function.

`LDAP_CALL` and `LDAP_CALLBACK` are used to set up calling conventions (for example, Pascal calling conventions on Windows). These are defined in the `lber.h` header file.

Registering the Rebind Function

Once you have a function that follows this prototype, you need to register the rebind function. You can do this in one of the following two ways:

- Call the LDAP API function `ldap_set_rebind_proc()`, specifying your function and any data that you want passed as an argument.
- Call `ldap_set_option()` to set the `LDAP_OPT_REBIND_FN` option to your function and the `LDAP_OPT_REBIND_ARG` option to specify any arguments you want passed to your rebind function.

Both of these methods register your rebind function.

Setting Up an In-Memory Cache

The Netscape LDAP SDK for C includes functions that allow you to create an in-memory cache of search results for your client. When send a search request and receive results, the results are cached. The next time your client issues the same search request, the results are read from the cache.

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

1. Call the `ldap_memcache_init()` function to create a new `LDAPMemCache` structure, which is the cache. Pass the pointer to this structure to subsequent operations.
2. Call the `ldap_memcache_set()` function to associate the cache with an LDAP connection handle (an LDAP structure).

For example:

Code Example 5-13 Creating an in-memory cache

```
#include "ldap.h"
...
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
...
LDAP          *ld;
LDAPMemCache  *dircache;
char *matched_msg = NULL, *error_msg = NULL;
int rc;
...
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}
...
/* Create an in-memory cache. */
rc = ldap_memcache_init( 0, 0, NULL, NULL, &dircache );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_memcache_init: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}

/* Associate the cache with the connection. */
rc = ldap_memcache_set( ld, dircache );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_memcache_set: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
```

Code Example 5-13 Creating an in-memory cache

```
}
...
```

When a search request is cached, the search criteria is used as the key to the item in the cache. If you run the same search again, the results are read from the cache. If you alter the criteria (for example, specifying that you want all attributes returned instead of just the `uid` attribute), your client gets the results from the server rather than from the cache.

The cache periodically checks for expired items and removes them from the cache. If you are writing a multi threaded application and want to set up a separate thread to keep the cache up-to-date, you can call the `ldap_memcache_update()` function.

To remove items from the cache or flush the cache, call the `ldap_memcache_flush()` function. When you are done working with the cache, call the `ldap_memcache_destroy()` function.

Handling Failover

While performing an LDAP operation, if the LDAP client loses the connection with the server, the LDAP API library returns an `LDAP_SERVER_DOWN` or `LDAP_CONNECT_ERROR` result code.

To reconnect to the server, you can do one of the following:

- Free the current connection handle and create a new connection handle.

  For details, see “Creating a New Connection Handle.”

- Use the reconnect option (`LDAP_OPT_RECONNECT`) to connect to the server again with the same connection handle.

  You can use this option if you do not want to free the connection handle (for example, if multiple threads are sharing the same connection handle). For details, see “Using the Reconnect Option.”

Creating a New Connection Handle

Call the `ldap_unbind()` or `ldap_unbind_s()` function to free the existing connection handle (the `LDAP` structure), and then call the `ldap_init()` function to open and initialize a new connection. For example:

Code Example 5-14 Initializing a new connection handle

```
#include "ldap.h"
...
LDAP *ld;
int tries = 0, rc = 0;
...
do {
  /* Call a function that performs an LDAP operation
  (my_ldap_request_function() can be any of these functions,
  such as ldap_search_ext_s()) */
  rc = my_ldap_request_function( ld );

  /* Check to see if the connection was lost. */
  if ( rc != LDAP_SERVER_DOWN && rc != LDAP_CONNECT_ERROR ) {
    return( rc ); /* Return result code. */
  }

  /* If the connection was lost, free the handle. */
  ldap_unbind( ld );

  /* Create a new connection handle and
   attempt to bind again. */
  if (( ld = ldap_init( hostlist, port )) != NULL ) {
    ldap_simple_bind_s();

    /* Perform any other initialization
    work on the connection handle. */
  }
} while ( ld != NULL && ++tries < 2 );
...
```

The disadvantage of this approach is that you need free the LDAP connection handle, which can make it difficult to share connection handles between threads. If you do not want to free the connection handle, you can use the reconnect option instead, as described in “Using the Reconnect Option.”

Using the Reconnect Option

To reconnect to the server without freeing the connection handle (for example, if multiple threads need to share the same connection handle), call the `ldap_set_option()` function to set the `LDAP_OPT_RECONNECT` option to `LDAP_OPT_ON`. Do this right after calling `ldap_init()`:

Code Example 5-15 Reconnecting to an existing connection handle

```
#include "ldap.h"
...
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
...
LDAP *ld;
...
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Set the reconnect option. */
if ( ldap_set_option( ld, LDAP_OPT_RECONNECT, LDAP_OPT_ON ) == 0 ) {
  /* success */
  } else {
  /* failure */
}
...
```

After setting this option, if the connection to the LDAP server has been lost (if the LDAP API library returns an `LDAP_SERVER_DOWN` or `LDAP_CONNECT_ERROR` result code to your client), call the `ldap_simple_bind_s()` function to reestablish a connection to one of the hosts specified in the `ldap_init()` function call.

If your client is able to reconnect with the server, the `ldap_simple_bind_s()` function call issues an LDAP bind request to the server and returns the result.

Note that if the client has been successfully authenticated to the server using the current `LDAP` structure and if the connection to the LDAP server has not been lost, `ldap_simple_bind_s()` returns `LDAP_SUCCESS` without contacting the LDAP server.

The function is designed to do this in cases where more than one thread shares the same LDAP connection handle and receives an LDAP_SERVER_DOWN or LDAP_CONNECT_ERROR error. If multiple threads call the ldap_simple_bind_s() function, the function is designed so that only one of the threads actually issues the bind operation. For the other threads, the function returns LDAP_SUCCESS without contacting the LDAP server.

The important side effect to note is that if the LDAP_OPT_RECONNECT option is set, ldap_simple_bind_s() may return LDAP_SUCCESS without contacting the LDAP server. (The function only returns this if a bind with the DN was successfully performed in the past.)

The following section of code attempts to reconnect to the server if the client is disconnected from the server:

Code Example 5-16 Reconnecting to a server after a disconnect

```
#include "ldap.h"
...
LDAP *ld;
int tries = 0, rc = 0;
...
do {
  /* Call a function that performs an LDAP operation
  (my_ldap_request_function() can be any of these functions,
  such as ldap_search_ext_s()) */
  rc = my_ldap_request_function( ld );

  /* Check to see if the connection was lost. */
  if ( rc != LDAP_SERVER_DOWN && rc != LDAP_CONNECT_ERROR ) {
    return( rc ); /* Return the result code. */
  }

  /* If the connection was lost, call
  ldap_simple_bind_s() to reconnect. */
  if ( ldap_simple_bind_s( ld, dn, passwd ) != LDAP_SUCCESS ) {
    /* failure -- could not reconnect */
    /* remember that ld as bad */
    return( rc );
  }
} while ( ++tries < 2 );
```

Chapter 6

Searching the Directory

This chapter explains how to call the LDAP API functions to search the directory, retrieve search results, and get attributes and values from each entry in the search results. The chapter also provides examples of calling synchronous and asynchronous functions to search the directory.

This chapter contains the following sections, which explain how to create and execute a search of the directory:

- Overview: Searching with LDAP API Functions
- Sending a Search Request
- Sorting the Search Results
- Freeing the Results of a Search
- Example: Searching the Directory (Asynchronous)
- Example: Searching the Directory (Synchronous)
- Reading an Entry
- Listing Subentries

Overview: Searching with LDAP API Functions

In the LDAPv3 protocol, a server can send three different types of results back to the client:

- Directory entries found by the search.
- Any search references found within the scope of the search (a search reference is a reference to another LDAP server).

- An LDAP result code specifying the result of the search operation.

Results are represented by `LDAPMessage` structures.

Note that in order to receive search references from LDAPv3 servers (such as the iPlanet Directory Server), you must identify your client as an LDAPv3 client. If you do not, the server will return the LDAP error code `LDAP_PARTIAL_RESULTS` and a set of referrals. See "Specifying the LDAP Version of Your Client" for details.

The Netscape LDAP SDK for C provides functions that allow you to search the directory and retrieve results from the server:

- You can send a search request by calling the synchronous function `ldap_search_ext_s()` or the asynchronous function `ldap_search_ext()`.

  (For more information about the difference between synchronous and asynchronous functions, see "Calling Synchronous and Asynchronous Functions.")

  The server sends back matching entries or search references to your client.

- If you are retrieving the results one at a time, you can call `ldap_result()` to get each result (an `LDAPMessage` structure) and determine what type of result (entry or search reference) was sent from the server.

- If you are retrieving a chain of results, you can call `ldap_first_message()` and `ldap_next_message()` to iterate through the results in the chain.

  If you are just interested in entries, you can call `ldap_first_entry()` and `ldap_next_entry()`.

  If you are just interested in search references, you can call `ldap_first_reference()` and `ldap_next_reference()`.

- To get an entry from a result (an `LDAPMessage` structure), call `ldap_first_entry()`.

- To get a search reference from a result (an `LDAPMessage` structure), call `ldap_parse_reference()`.

- To get the LDAP result code for the search operation from a result (an `LDAPMessage` structure), call `ldap_parse_result()`.

For more information, refer to the following sections in this chapter: "Example: Searching the Directory (Synchronous)" and "Example: Searching the Directory (Asynchronous)".

Sending a Search Request

To search the directory, call `ldap_search_ext_s()` or `ldap_search_ext()`:

- `ldap_search_ext_s()` is a synchronous function. This function blocks until all results have been received from the server.
- `ldap_search_ext()` is an asynchronous function. This function sends an LDAP search request to the server. You can do other work while periodically checking to see if the server has returned any results.

These two functions are declared as follows:

```
LDAP_API(int) LDAP_CALL ldap_search_ext( LDAP *ld,
  const char *base, int scope, const char *filter,
  char **attrs, int attrsonly, LDAPControl **serverctrls,
  LDAPControl **clientctrls, struct timeval *timeoutp,
  int sizelimit, int *msgidp );

LDAP_API(int) LDAP_CALL ldap_search_ext_s( LDAP *ld,
  const char *base, int scope, const char *filter, char **attrs,
  int attrsonly, LDAPControl **serverctrls,
  LDAPControl **clientctrls, struct timeval *timeoutp,
  int sizelimit, LDAPMessage **res );
```

For either function, you specify the search criteria using the following parameters:

- `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) and whether you want to retrieve only the attribute type or the attribute type and its values

  For details, see "Specifying the Attributes to Retrieve."

You can also specify that you only want to return the names of attributes (and not the values) by passing a non-zero value for the `attrsonly` argument.

- `serverctrls` and `clientctrls` specify the LDAPv3 controls associated with this search operation

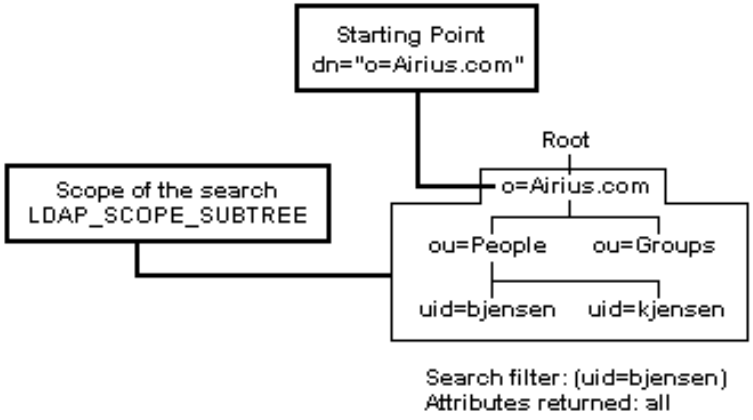
  For details on LDAPv3 controls, see Chapter 14, "Working with LDAP Controls."

- `timeoutp` and `sizelimit` specify the search constraints that you want applied to this search.

  For example, you can specify a different time-out period or maximum number of results that differ from the values of these options for the current session. See "Setting Search Preferences" for details.

Figure 6-1 illustrates how search criteria works.

Figure 6-1 Search criteria for an LDAP search operation



Specifying the Base DN and Scope

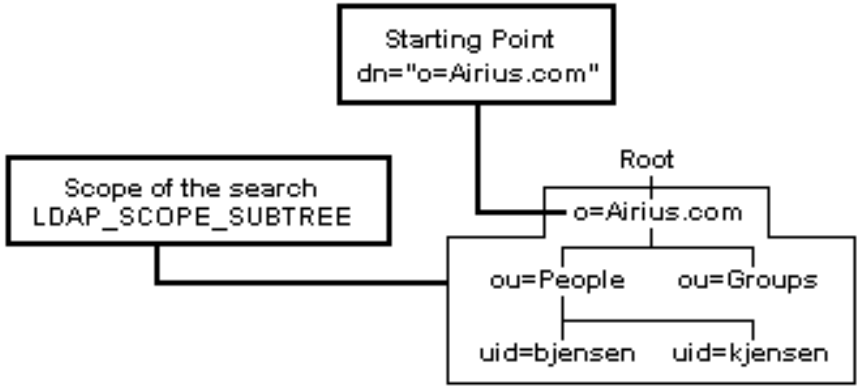
When sending a search request, you need to specify the base DN and scope of the search to identify the entries that you want searched.

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:

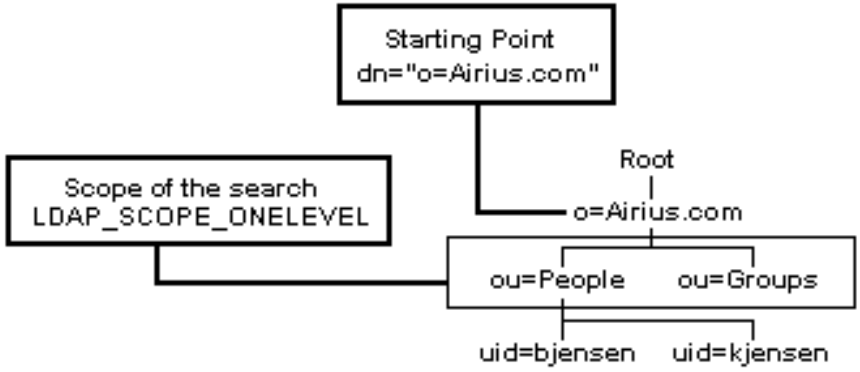
- LDAP_SCOPE_SUBTREE searches the base entry and all entries at all levels below the base entry (as illustrated in Figure 6-2).

Figure 6-2 Example of a search with the scope LDAP_SCOPE_SUBTREE



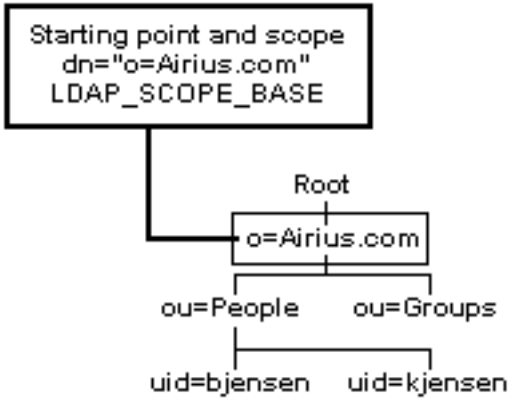
- LDAP_SCOPE_ONELEVEL searches all entries at one level below the base entry (as illustrated in Figure 6-3). The base entry is not included in the search. Use this setting if you just want a list of the entries under a given entry. (See "Listing Subentries" for an example.)

Figure 6-3 Example of a search with the scope LDAP_SCOPE_ONELEVEL



- LDAP_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" for an example.)

Figure 6-4 Example of a search with the scope LDAP_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 you are using.

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...`"

Table 6-1 Basic operators for search filters

Operator	Description	Example
<=	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.
|	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 (by passing the NULL-terminated array {"mail", "telephoneNumber", NULL} 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 LDAP_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 NULL-terminated array containing LDAP_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 specify the maximum number of results to be returned or the maximum amount of time to wait for a search. Use the `timeoutp` and `sizelimit` arguments of the `ldap_search_ext_s()` or `ldap_search_ext()` functions.

Note the following:

- To specify an infinite time limit (basically, no limit), create a `timeval` structure with `tv_sec = tv_usec = 0`, and pass a pointer to this as the `timeoutp` argument.
- To use the time limit specified by the `LDAP_OPT_TIMELIMIT` preference for this connection, pass `NULL` as the `timeoutp` argument.
- To specify an infinite size limit (basically, no limit), pass `LDAP_NO_LIMIT` as the `sizelimit` argument.

To specify these preferences for all searches under the current connection, call `ldap_set_option()` and set the `LDAP_OPT_SIZELIMIT` and `LDAP_OPT_TIMELIMIT` options. If you do not want to specify a limit (basically, no limit), set the value of the option to `LDAP_NO_LIMIT`.

Note that the LDAP server may already have time and size constraints set up that you cannot override.

The following example sets these session preferences so that a search returns no more than 100 entries and takes no more than 30 seconds.

Code Example 6-1 Setting session search preferences

```
#include <stdio.h>
#include "ldap.h"
...
LDAP *ld;
int max_ret, max_tim;
char *host = "localhost";
...
/* Initialize a session with the LDAP server ldap.netscape.com:389. */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Set the maximum number of entries returned. */
max_ret = 100;
ldap_set_option(ld, LDAP_OPT_SIZELIMIT, (void *)&max_ret );


/* Set the maximum number of seconds to wait. */
max_tim = 30;
ldap_set_option( ld, LDAP_OPT_TIMELIMIT, (void *)&max_tim );
...
```

Example of Sending a Search Request

The following example sends a search request to the server for all entries with the last name (or surname) "Jensen" in the airius.com organization.

Code Example 6-2 Sending a search request

```
#include <stdio.h>
#include "ldap.h"
...
#define BASEDN "o=airius.com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
...
LDAP      *ld;
int      msgid, rc;
...
/* Send the search request. */
rc = ldap_search_ext( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL,
  NULL, NULL, LDAP_NO_LIMIT, &msgid );
```

Code Example 6-2 Sending a search request

```
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}
...
```

Getting Search Results

In the LDAPv3 protocol, a server can return the following types of results to your client:

- Directory entries found by the search (those entries matching the search criteria).
- Any search references found within the scope of the search (a search reference is a reference to another LDAP server).
- An LDAP result code specifying the result of the search operation.

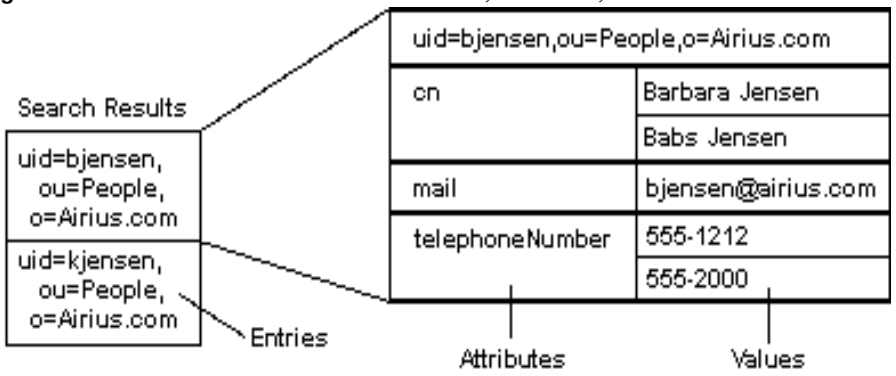
The server returns the search results as a chain of `LDAPMessage` structures. Each structure contains an entry, a search reference, or an LDAP result code. Because the results are represented as a chain, you should not free individual `LDAPMessage` structures within the chain. (When you are done working with the results, you can free the chain itself, rather than individual structures within the chain.)

To access data from entries found by the search, you need to follow this general process:

1. Get each entry in the results.
2. Get the attributes from each entry.
3. Get the values from each attribute.

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



This section explains how to get search results and retrieve data from the search results. Topics covered include:

- Getting Results Synchronously
- Getting Results Asynchronously
- Iterating Through a Chain of Results
- Getting Distinguished Names for Each Entry
- Getting Attributes from an Entry
- Getting the Values of an Attribute
- Getting Referrals from Search References

Getting Results Synchronously

If you called the `ldap_search_ext_s()` function to search the directory synchronously, the function blocks until all results have been received. The function returns a chain of the results in the `result` parameter (a handle to an `LDAPMessage` structure).

You can iterate through the results in this chain by calling different API functions. See the section “Iterating Through a Chain of Results” for details.

Getting Results Asynchronously

If you use the asynchronous function `ldap_search_ext()` instead, you need to call `ldap_result()` to determine if the server sent back any results:

```
LDAP_API(int) LDAP_CALL ldap_result( LDAP *ld, int msgid, int all,
  struct timeval *timeout, LDAPMessage **result );
```

You can specify how you want to get the results:

- To get the results one at a time (as the client receives them from the server), pass `LDAP_MSG_ONE` as the `all` argument.
- To get the results all at once (in other words, to block until all results are received), pass `LDAP_MSG_ALL` as the `all` argument.
- To get the results received so far, pass `LDAP_MSG_RECEIVED` as the `all` argument.

If you specify either `LDAP_MSG_ALL` or `LDAP_MSG_RECEIVED`, the function passes back a chain of search results as the `result` argument. For details on how to retrieve the results from this chain, see "Iterating Through a Chain of Results."

If specify `LDAP_MSG_ONE`, the function passes back a single search result as the `result` argument. The function normally returns the type of the first search result; in this case, since only one result is returned, the function returns the type of that result. A search result can be one of the following types:

To determine what type of result was returned, call the `ldap_msgtype()` function. A search result can be one of the following types:

- `LDAP_RES_SEARCH_ENTRY` indicates that the result is an entry found in the search.

  You can pass the `LDAPMessage` structure representing the entry to the `ldap_get_dn()` function to get the DN of the entry or the `ldap_first_attribute()` and `ldap_next_attribute()` functions to get the attributes of the entry.

  For details, see "Getting Distinguished Names for Each Entry" and "Getting Attributes from an Entry."

- `LDAP_RES_SEARCH_REFERENCE` indicates that the result is a search reference found within the scope of the search.

  You can pass the `LDAPMessage` structure representing the search reference to the `ldap_parse_reference()` function to get the referrals (LDAP URLs) to other servers.

For details, see “Getting Referrals from Search References.”

- `LDAP_RES_SEARCH_RESULT` indicates that the result is the final result sent by the server to indicate the result of the LDAP search operation.

  You can pass the `LDAPMessage` structure representing the result to the `ldap_parse_result()` function to get the LDAP result code for the search operation. (For a list of possible result codes for an LDAP search operation, see the result code documentation for the `ldap_search_ext_s()` function.)

  For details, see “Getting the Information from an LDAPMessage Structure”.

Note that in order to receive search references from an LDAPv3 server (such as the iPlanet Directory Server), you must identify your client as an LDAPv3 client. If you do not, the server will return the LDAP error code `LDAP_PARTIAL_RESULTS` and a set of referrals. See “Specifying the LDAP Version of Your Client” for details.

The following section of code gets results one at a time and checks the type of result:

Code Example 6-3 Retrieving search results

```
#include <stdio.h>
#include "ldap.h"
...
#define BASEDN "o=airius.com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
...
LDAP      *ld;
LDAPMessage    *res;
int        msgid, rc, parse_rc, finished = 0;
struct timeval  zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP search request. */
rc = ldap_search_ext( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
  NULL, LDAP_NO_LIMIT, &msgid );
...
/* Poll the server for the results of the search operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
  switch ( rc ) {
  case -1:
    /* An error occurred. */
    ...
  case 0:
    /* The timeout period specified by zerotime was exceeded. */
    ...
  case LDAP_RES_SEARCH_ENTRY:
    /* The server sent one of the entries found by the search. */
```

Code Example 6-3 Retrieving search results

```
    ...
  case LDAP_RES_SEARCH_REFERENCE:
    /* The server sent a search reference .*/
    ...
  case LDAP_RES_SEARCH_RESULT:
    /* Parse the final result received from the server. */
    ...
  }
...
}
...
```

Iterating Through a Chain of Results

The results of a search are represented by a chain of `LDAPMessage` structures. Each entry and search reference is contained in an `LDAPMessage` structure. The final result code of the LDAP search operation is also contained in one of these structures.

To retrieve results from a chain of search results, you can call one of the following sets of functions:

- To get each entry and search reference in the result, call `ldap_first_message()` and `ldap_next_message()`. Both of these functions return a pointer to an `LDAPMessage` structure that represents an entry, search reference, or LDAP result code.

  You can get the count of the results in the chain by calling `ldap_count_messages()`.

- If you just want to retrieve the entries from the chain, call `ldap_first_entry()` and `ldap_next_entry()`. Both of these functions return a pointer to an `LDAPMessage` structure that represents an entry.

  You can get the count of the entries in the chain by calling `ldap_count_entries()`.

- If you just want to retrieve the search references from the chain, call `ldap_first_reference()` and `ldap_next_reference()`. Both of these functions return a pointer to an `LDAPMessage` structure that represents a search reference.

  You can get the count of the search references in the chain by calling `ldap_count_references()`.

Note that each LDAPMessage structure is part of a chain and can point to other structures in the chain. You should not attempt to free individual LDAPMessage structures from memory; you may lose the rest of the results if you do this.

If you are iterating through each result, you can determine the type of the result by calling the ldap_msgtype() function. A search result can be one of the following types:

- LDAP_RES_SEARCH_ENTRY indicates that the result is an entry found in the search.

  You can pass the LDAPMessage structure representing the entry to the ldap_get_dn() function to get the DN of the entry or the ldap_first_attribute() and ldap_next_attribute() functions to get the attributes of the entry.

  For details, see "Getting Distinguished Names for Each Entry" and "Getting Attributes from an Entry."

- LDAP_RES_SEARCH_REFERENCE indicates that the result is a search reference found within the scope of the search.

  You can pass the LDAPMessage structure representing the search reference to the ldap_parse_reference() function to get the referrals (LDAP URLs) to other servers.

  For details, see "Getting Referrals from Search References."

- LDAP_RES_SEARCH_RESULT indicates that the result is the final result sent by the server to indicate the result of the LDAP search operation.

  You can pass the LDAPMessage structure representing the result to the ldap_parse_result() function to get the LDAP result code for the search operation.

  For details, see "Getting the Information from an LDAP Structure."

Note that in order to receive search references from an LDAPv3 server (such as the iPlanet Directory Server), you must identify your client as an LDAPv3 client. If you do not, the server will return the LDAP error code LDAP_PARTIAL_RESULTS and a set of referrals. See "Specifying the LDAP Version of Your Client" for details.

The following section of code retrieves each result in a chain and determines the type of the result.

Code Example 6-4 Retrieving a chain of results

```
#include <stdio.h>
#include "ldap.h"
...
#define BASEDN "o=airius.com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
...
LDAP      *ld;
LDAPMessage    *res, *msg;
BerElement    *ber;
char      *matched_msg = NULL, *error_msg = NULL;
int        rc, msgtype, num_entries = 0, num_refs = 0;
...
/* Perform the search operation. */
rc = ldap_search_ext_s( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
  NULL, LDAP_NO_LIMIT, &res );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
  if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
  }
  /* If the server cannot find an entry and
    returns the portion of the DN that can find
    an entry, print it out. (For details, see
    "Receiving the Portion of the DN Matching an Entry.") */
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
  ldap_unbind_s( ld );
  return( 1 );
}
...
num_entries = ldap_count_entries( ld, res );
num_refs = ldap_count_references( ld, res );
...
/* Iterate through the results. */
for ( msg = ldap_first_message( ld, res ); msg != NULL;
  msg = ldap_next_message( ld, msg ) ) {

  /* Determine what type of message was sent from the server. */
  msgtype = ldap_msgtype( msg );
  switch( msgtype ) {
  case LDAP_RES_SEARCH_ENTRY:
    /* The result is an entry. */
    ...
  case LDAP_RES_SEARCH_REFERENCE:
    /* The result is a search reference. */
    ...
  case LDAP_RES_SEARCH_RESULT:
    /* The result is the final result sent by the server. */
    ...
```

Code Example 6-4 Retrieving a chain of results

```
  }
...
}
...
```

Getting Distinguished Names for Each Entry

Because the distinguished name of an entry differentiates it from other entries, you may want to access the distinguished names of entries in the search results. You may also want to parse the name into its individual components.

The LDAP API provides functions for both of these tasks. You can call functions to get the name of an individual entry and to split the name into its components.

Getting the Distinguished Name of an Entry

To get the distinguished name of an entry, call the `ldap_get_dn()` function. The function returns the distinguished name of the entry.

When you are finished working with the distinguished name returned by this function, you should free it from memory by calling the `ldap_memfree()` function.

Code Example 6-5 prints the distinguished name for each entry found in a search: Obtaining the distinguished names for entries returned by a search.

Code Example 6-5 Obtaining the distinguished names for entries returned by a search

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
char *dn;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
      NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* For each matching entry found, print the name of the entry.*/
for ( e = ldap_first_entry( ld, result ); e != NULL;
      e = ldap_next_entry( ld, e ) ) {
```

Code Example 6-5 Obtaining the distinguished names for entries returned by a search

```
  if ( ( dn = ldap_get_dn( ld, e ) ) != NULL ) {
    printf( "dn: %s\n", dn );
    /* Free the memory used for the DN when done */
    ldap_memfree( dn );
  }
}
/* Free the result from memory when done. */
ldap_msgfree( result );
```

Getting the Components of a Distinguished Name

If you want to access individual components of a distinguished name or a relative distinguished name, call the `ldap_explode_dn()` and `ldap_explode_rdn()` functions.

Both functions return a `NULL`-terminated array of the components of the distinguished name. When you are done working with this array, you should free it by calling the `ldap_value_free()` function.

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 `0` if you want to include component names in the array:

  ```
  ldap_explode_dn( "uid=bjensen, ou=People, o=airius.com", 0 )
  ```

  The function returns this array:

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

- Set `notypes` to `1` if you don't want to include the component names in the array:

  ```
  ldap_explode_dn( "uid=bjensen, ou=People, o=airius.com", 1 )
  ```

  The function returns this array:

  ```
  { "bjensen", "People", "airius.com", NULL }
  ```

Getting Attributes from an Entry

To get the value of the first attribute in an entry, call the `ldap_first_attribute()` function.

This function returns the name of the first attribute in the entry. To get the value of this attribute, you need to pass the attribute name to the `ldap_get_values()` or `ldap_get_values_len()` functions. (See “Getting the Values of an Attribute” for details.)

To get the name of the next attribute, call the `ldap_next_attribute()` function.

Note that operational attributes such as `creatorsName` and `modifyTimestamp` are not normally returned in search results unless you explicitly specify them by name in the search request. For more information, see “Specifying the Attributes to Retrieve.”

When you are finished iterating through the attributes, you need to free the `BerElement` structure allocated by the `ldap_first_attribute()` function, if the structure is not `NULL`. To free this structure, call the `ldap_ber_free()` function.

You should also free the attribute name returned by the `ldap_first_attribute()` function. To free the attribute name, call the `ldap_memfree()` function.

The following section of code retrieves each attribute of an entry:

Code Example 6-6 Retrieving entry attributes

```
#include <stdio.h>
#include <ldap.h>
...
LDAP         *ld;
LDAPMessage  *result, *e;
BerElement   *ber;
char         *a;
char         *my_searchbase = "o=airius.com";
char         *my_filter = "(sn=Jensen)";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
      NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Retrieve the attributes of the entry. */
  for (a = ldap_first_attribute(ld, e, &ber); a != NULL;
        a = ldap_next_attribute(ld, e, ber)){
    ...
    /* Code to get and manipulate attribute values */
    ...
    }
    ldap_memfree( a );
  }
```

Code Example 6-6 Retrieving entry attributes

```
  /* Free the BerElement structure from memory when done. */
  if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
  }
...
```

Getting the Values of an Attribute

The values of an attribute are represented by a `NULL`-terminated array. The values are either a list of strings (if the attribute contains string data, such as a name or phone number) or a list of `berval` structures (if the attribute contains binary data, such as a JPEG file or an audio file).

- To get the values of an attribute that contains string data, call the `ldap_get_values()` function.

  The `ldap_get_values()` function returns a `NULL`-terminated array of strings representing the value of the attribute.

- To get the values of an attribute that contains binary data, call the `ldap_get_values_len()` function.

  The `ldap_get_values_len()` function returns a `NULL`-terminated array of `berval` structures representing the value of the attribute.

To get the number of values in an attribute, call the `ldap_count_values()` or `ldap_count_values_len()` function. Both functions return the number of values in the attribute.

When you have finished working with the values of the attribute, you need to free the values from memory. To do this, call the `ldap_value_free()` or `ldap_value_free_len()` function.

The following section of code gets and prints the values of an attribute in an entry. This example assumes that all attributes have string values:

Code Example 6-7 Retrieving attribute values

```
#include <stdio.h>
#include <ldap.h>
...
LDAP          *ld;
LDAPMessage   *result, *e;
BerElement    *ber;
```

Code Example 6-7 Retrieving attribute values

```
char          *a;
char          **vals;
char            *my_searchbase = "o=airius.com";
char            *my_filter = "(sn=Jensen)";
int i;
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
      NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Get the first matching attribute. */
a = ldap_first_attribute( ld, e, &ber );

/* Get the values of the attribute. */
if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
  for ( i = 0; vals[i] != NULL; i++ ) {
    /* Print the name of the attribute and each value */
    printf( "%s: %s\n", a, vals[i] );
  }
  /* Free the attribute values from memory when done. */
  ldap_value_free( vals );
}
...
```

The following section of code gets the first value of the `jpegPhoto` attribute and saves the JPEG data to a file:

Code Example 6-8 Getting and saving the first attribute value

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a;
struct berval photo_data;
struct berval **list_of_photos;
FILE *out;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory. */
```

Code Example 6-8 Getting and saving the first attribute value

```
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
      NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Find the jpegPhoto attribute. */
a = ldap_first_attribute( ld, e, &ber );
while ( strcasecmp( a, "jpegphoto" ) != 0 ) {
  a = ldap_next_attribute( ld, e, ber );
}

/* Get the value of the attribute. */
if ( ( list_of_photos = ldap_get_values_len( ld, e, a ) ) != NULL ) {
  /* Prepare to write the JPEG data to a file */
  if ( ( out = fopen( "photo.jpg", "wb" ) ) == NULL ) {
    perror( "fopen" );
    return( 1 );
  }
  /* Get the first JPEG. */
  photo_data = *list_of_photos[0];
  /* Write the JPEG data to a file */
  fwrite( photo_data.bv_val, photo_data.bv_len, 1, out );
  fclose( out );
  /* Free the attribute values from memory when done. */
  ldap_value_free_len( list_of_photos );
}
...
```

Getting Referrals from Search References

A search reference returned from the server contains one or more referrals (LDAP URLs that identify other LDAP servers). To get these referrals, you need to call the `ldap_parse_reference()` function.

The following section of code gets and prints the referrals in a search reference:

Code Example 6-9 Obtaining a referral

```
#include <stdio.h>
#include "ldap.h"
...
LDAP          *ld;
LDAPMessage   *msg;
```

Code Example 6-9 Obtaining a referral

```
char          **referrals;
int           i, rc, parse_rc;
...
parse_rc = ldap_parse_reference( ld, msg, &referrals, NULL, 0 );
if ( parse_rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_parse_result: %s\n",
    ldap_err2string( parse_rc ) );
  ldap_unbind( ld );
  return( 1 );
}
if ( referrals != NULL ) {
  for ( i = 0; referrals[ i ] != NULL; i++ ) {
    printf( "Search reference: %s\n\n", referrals[ i ] );
  }
  ldap_value_free( referrals );
}
...
```

Sorting the Search Results

The LDAP API contains functions that you can use to sort entries and values in the search results. You can either sort entries on your client, or you can specify that the server should return sorted results.

To sort results on the server, you need to send a server-side sorting control with the search request. For details, see “Using the Server-Side Sorting Control.”

The rest of this section explains how to sort results on your client. Topics include:

- Sorting Entries by an Attribute
- Sorting Entries by Multiple Attributes
- Sorting the Values of an Attribute

Note that you need to return any attributes that you plan to use for sorting the results. For example, if you plan to sort the results by email address, make sure that the mail attribute is one of the attributes returned in the search. For details on specifying the attributes returned, see “Specifying the Attributes to Retrieve.”

Sorting Entries by an Attribute

To sort the search results by a particular attribute, call the `ldap_sort_entries()` function. Note that if you don't specify an attribute for sorting (that is, if you pass `NULL` for the `attr` parameter), the entries are sorted by DN.

The following section of code sorts entries by the `roomNumber` attribute:

Code Example 6-10 Sorting entries by attribute value

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
char *sortby = "roomNumber";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Sort the results by room number, using strcasecmp. */
if (ldap_sort_entries(ld, &result, sortby, strcasecmp) != LDAP_SUCCESS){
  ldap_perror( ld, "ldap_sort_entries" );
  return( 1 );
}
...
```

Sorting Entries by Multiple Attributes

To sort the search results by a particular attribute, call the `ldap_multisort_entries()` function. Note that if you don't specify a set of attributes for sorting (that is, if you pass `NULL` for the `attr` parameter), the entries are sorted by DN.

The following section of code sorts entries first by the `roomNumber` attribute, then by the `telephoneNumber` attribute.

Code Example 6-11 Sorting entries by multiple values

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
LDAP *ld;
LDAPMessage *res;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
char *attrs[2];
attrs[0] = “roomNumber”;
attrs[1] = “telephoneNumber”;
attrs[2] = NULL;
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
      NULL, 0, &res ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Sort the results, using strcasecmp. */
if (ldap_multisort_entries(ld,&res,attrs, strcasecmp) != LDAP_SUCCESS){
  ldap_perror( ld, "ldap_sort_entries" );
  return( 1 );
}
```

Sorting the Values of an Attribute

You can also sort the values of a particular attribute. To do this, call the `ldap_sort_values()` function.

In the `ldap_sort_values()` function, the comparison function must pass parameters of the type `char **`. You should use the `ldap_sort_strcasecmp()` function, rather than a function like `strcasecmp()` (which passes parameters of the type `char *`).

The following section of code sorts the values of attributes before printing them.

Code Example 6-12 Sorting attribute values

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a, *dn;
```

Code Example 6-12 Sorting attribute values

```
char **vals;
int i;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
...
    if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
      /* Sort the values of the attribute */
      if (ldap_sort_values(ld, vals, strcasecmp)) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_sort_values" );
        return( 1 );
      }
      /* Print the values of the attribute. */
      for ( i = 0; vals[i] != NULL; i++ ) {
        printf( "%s: %s\n", a, vals[i] );
      }
      /* Free the values from memory. */
      ldap_value_free( vals );
    }
...
```

Freeing the Results of a Search

The results of the search are returned in an `LDAPMessage` structure. After you are done working with the search results, you should free this structure from memory.

To free the search results, call the `ldap_msgfree()` function. The `ldap_msgfree()` function returns the type of the last message freed from memory.

Example: Searching the Directory (Asynchronous)

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

Code Example 6-13 Asynchronous searching

```
#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
```

Code Example 6-13 Asynchronous searching

```
#define BASEDN "o=airius.com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
int
main( int argc, char **argv )
{
    LDAP           *ld;
    LDAPMessage    *res;
    BerElement     *ber;
  LDAPControl      **serverctrls;
    char           *a, *dn, *matched_msg = NULL, *error_msg = NULL;
    char           **vals, **referrals;
    int           version, i, msgid, rc, parse_rc, finished = 0,
                   num_entries = 0, num_refs = 0;
    struct         timeval  zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
    }
  version = LDAP_VERSION3;
  if ( ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version ) !=
LDAP_SUCCESS ) {
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_set_option: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
    /* Bind to the server anonymously. */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    /* If the server cannot find an entry,
      print the portion of the DN that matches
      an existing entry. (For details, see
      "Receiving the Portion of the DN Matching an Entry.") */
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
    }
    /* Send the LDAP search request. */
   rc = ldap_search_ext( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL, NULL,
LDAP_NO_LIMIT, &msgid );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
```

Code Example 6-13 Asynchronous searching

```
    ldap_unbind( ld );
    return( 1 );
    }
    /* Poll the server for the results of the search operation.
     Passing LDAP_MSG_ONE indicates that you want to receive
     the entries one at a time, as they come in.  If the next
     entry that you retrieve is NULL, there are no more entries. */
    while ( !finished ) {
    rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
    /* The server can return three types of results back to the client,
       and the return value of ldap_result() indicates the result type:
       LDAP_RES_SEARCH_ENTRY identifies an entry found by the search,
       LDAP_RES_SEARCH_REFERENCE identifies a search reference returned
       by the server, and LDAP_RES_SEARCH_RESULT is the last result
       sent from the server to the client after the operation completes.
       You need to check for each of these types of results. */
    switch ( rc ) {
    case -1:
      /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      return( 1 );
    case 0:
      /* The timeout period specified by zerotime was exceeded.
         This means that the server has still not yet sent the
         results of the search operation back to your client.
         Break out of this switch statement, and continue calling
         ldap_result() to poll for results. */
      break;
    case LDAP_RES_SEARCH_ENTRY:
      /* The server sent one of the entries found by the search
         operation. Print the DN, attributes, and values of the entry. */
      /* Keep track of the number of entries found. */
      num_entries++;
      /* Get and print the DN of the entry. */
      if (( dn = ldap_get_dn( ld, res )) != NULL ) {
        printf( "dn: %s\n", dn );
        ldap_memfree( dn );
      }
      /* Iterate through each attribute in the entry. */
      for ( a = ldap_first_attribute( ld, res, &ber );
      a != NULL; a = ldap_next_attribute( ld, res, ber ) ) {
        /* Get and print all values for each attribute. */
        if (( vals = ldap_get_values( ld, res, a )) != NULL ) {
          for ( i = 0; vals[ i ] != NULL; i++ ) {
            printf( "%s: %s\n", a, vals[ i ] );
          }
          ldap_value_free( vals );
        }
        ldap_memfree( a );
      }
      if ( ber != NULL ) {
        ber_free( ber, 0 );
      }
```

Code Example 6-13 Asynchronous searching

```
      printf( "\n" );
      ldap_msgfree( res );
      break;
    case LDAP_RES_SEARCH_REFERENCE:
      /* The server sent a search reference encountered during the
         search operation. */
      /* Keep track of the number of search references returned from
         the server. */
      num_refs++;
      /* Parse the result and print the search references.
         Ideally, rather than print them out, you would follow the
         references. */
      parse_rc = ldap_parse_reference( ld, res, &referrals, NULL, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      if ( referrals != NULL ) {
        for ( i = 0; referrals[ i ] != NULL; i++ ) {
          printf( "Search reference: %s\n\n", referrals[ i ] );
        }
        ldap_value_free( referrals );
      }
      break;
    case LDAP_RES_SEARCH_RESULT:
      /* Parse the final result received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      finished = 1;
      parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
NULL, &serverctrls, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      /* Check the results of the LDAP search operation. */
      if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
          fprintf( stderr, "Part of the DN that matches an existing entry:
%s\n", matched_msg );
        }
      } else {
        printf( "Search completed successfully.\n"
          "Entries found: %d\n"
```

Code Example 6-13 Asynchronous searching

```
          "Search references returned: %d\n"
          "Counted to %d while waiting for the search operation.\n",
          num_entries, num_refs, global_counter );
      }

      break;

    default:
      break;
    }

    /* Do other work here while waiting for the search operation to complete.
*/
    if ( !finished ) {
      do_other_work();
    }
  }
    /* Disconnect when done. */
    ldap_unbind( ld );
    return( 0 );
}
/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
static void
do_other_work()
{
    global_counter++;
}
```

Example: Searching the Directory (Synchronous)

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

Code Example 6-14 Synchronous searching

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BASEDN "o=airius.com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
int
main( int argc, char **argv )
```

Code Example 6-14 Synchronous searching

```
{
    LDAP           *ld;
    LDAPMessage    *res, *msg;
  LDAPControl      **serverctrls;
    BerElement     *ber;
    char           *a, *dn, *matched_msg = NULL, *error_msg = NULL;
    char           **vals, **referrals;
    int            version, i, rc, parse_rc, msgtype, num_entries = 0,
                     num_refs = 0;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
    }
  version = LDAP_VERSION3;
  if ( ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version ) !=
        LDAP_SUCCESS ) {
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_set_option: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
    /* Bind to the server anonymously. */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr, "Part of the DN that matches an existing entry: %s\n",
matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
    }
    /* Perform the search operation. */
    rc = ldap_search_ext_s( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
NULL, LDAP_NO_LIMIT, &res );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr, "Part of the DN that matches an existing entry: %s\n",
matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
    }
    num_entries = ldap_count_entries( ld, res );
  num_refs = ldap_count_references( ld, res );
```

Code Example 6-14 Synchronous searching

```
  /* Iterate through the results. An LDAPMessage structure sent back from
      a search operation can contain either an entry found by the search,
      a search reference, or the final result of the search operation. */
  for ( msg = ldap_first_message( ld, res ); msg != NULL; msg =
ldap_next_message( ld, msg ) ) {
    /* Determine what type of message was sent from the server. */
    msgtype = ldap_msgtype( msg );
    switch( msgtype ) {
    /* If the result was an entry found by the search, get and print the
       attributes and values of the entry. */
    case LDAP_RES_SEARCH_ENTRY:
      /* Get and print the DN of the entry. */
      if (( dn = ldap_get_dn( ld, res )) != NULL ) {
        printf( "dn: %s\n", dn );
        ldap_memfree( dn );
      }
      /* Iterate through each attribute in the entry. */
      for ( a = ldap_first_attribute( ld, res, &ber );
      a != NULL; a = ldap_next_attribute( ld, res, ber ) ) {
        /* Get and print all values for each attribute. */
        if (( vals = ldap_get_values( ld, res, a )) != NULL ) {
          for ( i = 0; vals[ i ] != NULL; i++ ) {
            printf( "%s: %s\n", a, vals[ i ] );
          }
          ldap_value_free( vals );
        }
        ldap_memfree( a );
      }
      if ( ber != NULL ) {
        ber_free( ber, 0 );
      }
      printf( "\n" );
      break;
    case LDAP_RES_SEARCH_REFERENCE:
      /* The server sent a search reference encountered during the
         search operation. */
      /* Parse the result and print the search references.
         Ideally, rather than print them out, you would follow the
         references. */
      parse_rc = ldap_parse_reference( ld, msg, &referrals, NULL, 0 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      if ( referrals != NULL ) {
        for ( i = 0; referrals[ i ] != NULL; i++ ) {
          printf( "Search reference: %s\n\n", referrals[ i ] );
        }
        ldap_value_free( referrals );
      }
      break;
    case LDAP_RES_SEARCH_RESULT:
```

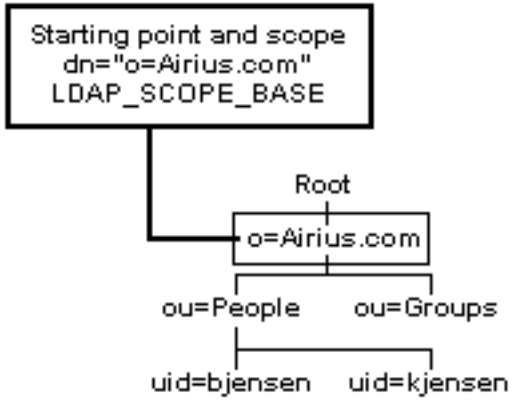
Code Example 6-14 Synchronous searching

```
      /* Parse the final result received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      parse_rc = ldap_parse_result( ld, msg, &rc, &matched_msg, &error_msg,
NULL, &serverctrls, 0 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      /* Check the results of the LDAP search operation. */
      if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
          fprintf( stderr, "Part of the DN that matches an existing entry:
%s\n", matched_msg );
        }
      } else {
        printf( "Search completed successfully.\n"
          "Entries found: %d\n"
          "Search references returned: %d\n",
          num_entries, num_refs );
      }

      break;

    default:
      break;
    }
  }
    /* Disconnect when done. */
    ldap_unbind( ld );
    return( 0 );
}
```

Reading an Entry

You can use the search functions to read a specific entry in the directory. To read an entry, set the starting point of the search to the entry, and set the scope of the search to `LDAP_SCOPE_BASE` and specify `(objectclass=*)` for the search filter.

Figure 6-6 Using the LDAP_SCOPE_BASE scope to read an entry



The following section of code prints the attributes of the entry for Barbara Jensen.

Code Example 6-15 Obtaining a specific entry

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORT_NUMBER  LDAP_PORT
#define FIND_DN "uid=bjensen, ou=People, o=airius.com"
int
main( int argc, char **argv )
{
    LDAP  *ld;
    LDAPMessage  *result, *e;
    BerElement  *ber;
    char  *a;
    char  **vals;
    int    i, rc;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORT_NUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
    }
    /* Bind anonymously to the LDAP server. */
    if ( ( rc = ldap_simple_bind_s( ld, NULL, NULL ) ) != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    return( 1 );
    }
    /* Search for the entry. */
    if ( ( rc = ldap_search_ext_s( ld, FIND_DN, LDAP_SCOPE_BASE,
"(objectclass=*)",
```

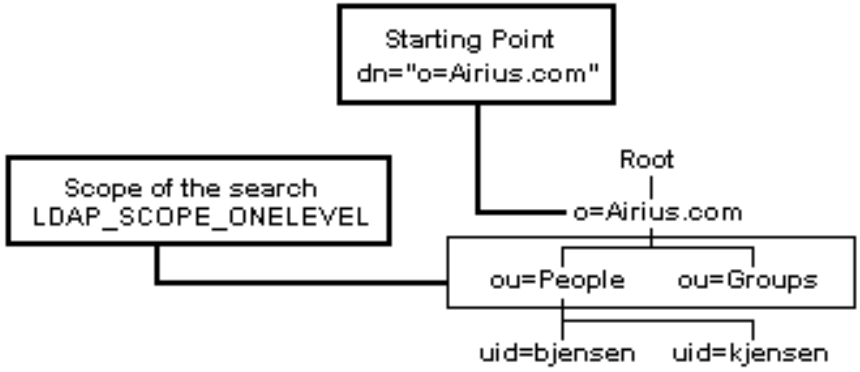
Code Example 6-15 Obtaining a specific entry

```
    NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result ) ) !=
LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    return( 1 );
    }
    /* Since we are doing a base search, there should be only one matching
entry.  */
    e = ldap_first_entry( ld, result );
  if ( e != NULL ) {
    printf( "\nFound %s:\n\n", FIND_DN );
    /* Iterate through each attribute in the entry. */
    for ( a = ldap_first_attribute( ld, e, &ber );
      a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
      /* For each attribute, print the attribute name and values. */
      if ((vals = ldap_get_values( ld, e, a)) != NULL ) {
        for ( i = 0; vals[i] != NULL; i++ ) {
          printf( "%s: %s\n", a, vals[i] );
        }
        ldap_value_free( vals );
      }
      ldap_memfree( a );
    }
    if ( ber != NULL ) {
      ber_free( ber, 0 );
    }
  }
  ldap_msgfree( result );
  ldap_unbind( ld );
  return( 0 );
}
```

Listing Subentries

You can use the search functions to list the subentries under a specific entry in the directory. To list the subentries, set the starting point of the search to the entry, and set the scope of the search to `LDAP_SCOPE_ONELEVEL`.

Figure 6-7 Using the LDAP_SCOPE_ONELEVEL scope to list subentries



The following section of code lists all entries one level below the `o=airius.com` entry in the directory hierarchy.

Code Example 6-16 Listing subentries

```
#include <stdio.h>
#include <ldap.h>

LDAP        *ld;
LDAPMessage *result, *e;
BerElement  *ber;
char        *a, *dn;
char        **vals;
char        *my_searchbase = "o=airius.com";
char        *my_filter = "(objectclass=*)"

/* Search one level under the starting point. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_ONELEVEL, my_filter,
      NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}
/* For each matching entry, print the entry name and its attributes. */
for ( e = ldap_first_entry( ld, result ); e != NULL;
      e = ldap_next_entry( ld, e ) ) {
  if ( ( dn = ldap_get_dn( ld, e ) ) != NULL ) {
    printf( "dn: %s\n", dn );
    ldap_memfree( dn );
  }
  for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
      a = ldap_next_attribute( ld, e, ber ) ) {
    if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
      for ( i = 0; vals[i] != NULL; i++ ) {
        printf( "%s: %s\n", a, vals[i] );
      }
```

Code Example 6-16 Listing subentries

```
      ldap_value_free( vals );
    }
    ldap_memfree( a );
  }
  if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
  }
  printf( "\n" );
}
ldap_msgfree( result );
...
```

Using Filter Configuration Files

This chapter explains how to use API functions 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
- Freeing Filters from Memory
- Creating Filters Programmatically

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 find and select 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  [scope]
  filter1-2   desc1-2                           [scope]
  pattern2    delimiters    filter2-1  desc2-1  [scope]
...
```

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) ...
  ```

  When you call functions like `ldap_getfirstfilter()` to retrieve a filter, 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.

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*

  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*

  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*-

  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 load a filter configuration file, call the ldap_init_getfilter() function.

You can also read the filter configuration file from a buffer in memory by calling the ldap_init_getfilter_buf() function.

Both functions return a pointer to an LDAPFiltDesc structure, which contains information about the filter. If an error occurs, both functions return NULL.

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 of `person` entries.

To retrieve the first filter that matches the search criteria, call the `ldap_getfirstfilter()` function.

To get the next filter that matches the search criteria, call the `ldap_getnextfilter()` function.

Both functions return a pointer to an `LDAPFiltInfo` structure, which contains information about the filter.

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"
```

This section of code retrieves filters that match the search criteria.

Code Example 7-1 Retrieving configuration filters

```
#include <stdio.h>
#include <ldap.h>
...
LDAP          *ld;
LDAPMessage   *result, *e;
BerElement    *ber;
char          *a, *dn;
char          **vals;
int i;
LDAPFiltDesc *ldfp;
LDAPFiltInfo *ldfi;
char buf[ 80 ]; /* contains the search criteria */
int found;
...
/* Load the filter configuration file into an LDAPFiltDesc structure. */
if ( ( ldfp = ldap_init_getfilter( "myfilters.conf" ) ) == NULL ) {
  perror( "Cannot open filter configuration file" );
}

/* Select a filter to use when searching for the value in buf.
Use filters under the "people" tag in the filter configuration file. */
found = 0;
```

Code Example 7-1 Retrieving configuration filters

```
for ( ldfi = ldap_getfirstfilter( ldfp, "people", buf ); ldfi != NULL;
  ldfi = ldap_getnextfilter( ldfp ) ) {

  /* Use the selected filter to search the directory. */
  if ( ldap_search_s( ld, "o=airius.com", ldfi->lfi_scope,
   ldfi->lfi_filter, NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
  } else {

    /* Once a filter gets results back, stop iterating through
    the different filters. */
    if ( ( found = ldap_count_entries( ld, result ) > 0 ) ) {
      break;
    } else {
      ldap_msgfree( result );
    }
  }
}

if ( found == 0 ) {
  printf( "No matching entries found.\n" );
} else {
  printf( "Found %d match%s where %s \"%s\"\n\n", found,
   found == 1 ? "" : "es", ldfi->lfi_desc, buf );
}

ldap_msgfree( result );
ldap_getfilter_free( ldfp );
...
```

Suppose that the search criteria is `bjensen@airius.com` and the client application finds one matching entry. In this case, the application prints the following output:

```
Found 1 match where email address is 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:

```
(&(objectClass=person)
```

Note that this filter now requires a suffix ")" to balance the number of parentheses. This prefix is automatically added to any filter retrieved through the `ldap_getfirstfilter()` and `ldap_getnextfilter()` functions. (See “Retrieving Filters” for details.) For example, suppose you use this filter in a filter configuration file:

```
(cn=Babs Jensen)
```

If you retrieve this filter through the `ldap_getfirstfilter()` or `ldap_getnextfilter()` function, you get the following filter:

```
(&(objectClass=person)(cn=Babs Jensen))
```

To add a prefix and suffix automatically to all filters retrieved from the filter configuration file, call the `ldap_set_filter_additions()` function.

The following section of code loads the filter configuration file named `myfilters.conf` into memory and adds the prefix `(&(objectClass=person)` and the suffix “`)`” to each filter retrieved:

Code Example 7-2 Adding prefixes and suffixes

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
char *prefix = "(&(objectClass=person)";
char *suffix = ")";
...
lfdp = ldap_init_getfilter( filter_file );
ldap_setfilteraffixes( lfdp, prefix, suffix );
...
```

Freeing Filters from Memory

When you complete your search, you should free the `LDAPFiltDesc` structure from memory. To free the `LDAPFiltDesc` structure, call the `ldap_getfilter_free()` function.

The following section of code frees the `LDAPFiltDesc` structure from memory after all searches are completed.

Code Example 7-3 Freeing filters from memory

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
...
/* Read the filter configuration file into an LDAPFiltDesc structure. */
lfdp = ldap_init_getfilter( filter_file );
...
/* Retrieve filters and perform searches. */
...
/* Free the configuration file (the LDAPFiltDesc structure). */
ldap_getfilter_free( lfdp );
...
```

Creating Filters Programmatically

You can build your own filters by calling the `ldap_create_filter()` function.

The following section of code builds the filter (`mail=bjensen@airius.com`).

Code Example 7-4 Creating filters

```
char buf[LDAP_FILT_MAXSIZ];
char *pattern = "(%a=%v);
char *attr = "mail";
char *value = "bjensen@airius.com";
...
ldap_create_filter( buf, LDAP_FILT_MAXSIZ, pattern, NULL, NULL, attr,
    value, NULL );
...
```

Chapter 8

Adding, Updating, and Deleting Entries

This chapter explains how to use the API functions to add, modify, delete, and rename entries in the directory. The chapter also provides examples of calling synchronous and asynchronous functions to perform these operations.

The chapter includes the following sections:

Specifying a New or Modified Attribute

In order to add or modify an entry in the directory, you need to specify information about the entry's attributes:

- If you are adding an entry, you must specify the attributes in the new entry.
- If you are modifying an entry, you must specify the attributes and values that you are adding, replacing, or removing.

In most of the cases above, you need to specify the following attribute information:

- The operation that you are performing, if you are modifying an existing entry (in other words, adding, modifying, or deleting the attribute in the existing entry).
- The type of attribute that you are working with (for example, the `sn` attribute or the `telephoneNumber` attribute).
- The values that you are adding or replacing in the attribute

To specify this information, you use an `LDAPMod` structure.

Code Example 8-1 Adding or modifying an attribute

```
typedef struct ldapmod {
  int mod_op;
  char *mod_type;
  union {
    char **modv_strvals;
    struct berval **modv_bvals;
  } mod_vals;
#define mod_values        mod_vals.modv_strvals
#define mod_bvalues       mod_vals.modv_bvals
} LDAPMod;
```

The following table details the fields in the LDAPMod data structure:

Table 8-1 LDAPMod field descriptions

Field	Description
mod_op	The operation to be performed on the attribute and the type of data specified as the attribute values. This field can have one of the following values: • LDAP_MOD_ADD adds a value to the attribute. • LDAP_MOD_DELETE removes the value from the attribute. • LDAP_MOD_REPLACE replaces all existing values of the attribute. In addition, if you are specifying binary values in the mod_bvalues field, you should use the bitwise OR operator (|) to combine LDAP_MOD_BVALUES with the operation type. For example: `mod->mod_op = LDAP_MOD_ADD | LDAP_MOD_BVALUES` Note that if you are using the structure to add a new entry, you can specify 0 for the mod_op field (unless you are adding binary values and need to specify LDAP_MOD_BVALUES). See "Adding a New Entry" for details.
mod_type	The attribute type that you want to add, delete, or replace the values of (for example, "sn" or "telephoneNumber").
mod_values	A pointer to a NULL-terminated array of string values for the attribute.
mod_bvalues	A pointer to a NULL-terminated array of berval structures for the attribute.

Note the following:

- If you specify LDAP_MOD_DELETE in the mod_op field and you remove all values in an attribute, the attribute is removed from the entry.
- If you specify LDAP_MOD_DELETE in the mod_op field and NULL in the mod_values field, the attribute is removed from the entry.
- If you specify LDAP_MOD_REPLACE in the mod_op field and NULL in the mod_values field, the attribute is removed from the entry.
- If you specify LDAP_MOD_REPLACE in the mod_op field and the attribute does not exist in the entry, the attribute is added to the entry.
- If you specify LDAP_MOD_ADD in the mod_op field and the attribute does not exist in the entry, the attribute is added to the entry.

If you've allocated memory for the structures yourself, you should free the structures when you're finished by calling the `ldap_mods_free()` function.

For information on how to set up `LDAPMod` structures to add and modify entries in the directory, see the sections "Specifying the Attributes" and "Specifying the Changes."

Adding a New Entry

To add an entry to the directory:

1. Create an array of `LDAPMod` structures to represent the attributes in the entry. Use the `LDAPMod` structure to specify the name and values of each attribute.
2. Call the `ldap_add_ext()` or `ldap_add_ext_s()`, passing in the array of `LDAPMod` structures and a distinguished name for the entry.

After you are done, call the `ldap_mods_free()` function to free any `LDAPMod` structures you've allocated.

Specifying the Attributes

This section describes the process of specifying the attributes and values of the new entry in the following sections:

- Specifying a Single Value in an Attribute
- Specifying Multiple Values in an Attribute
- Specifying Binary Data as the Values of an Attribute
- Specifying the Attributes in the Entry

Specifying a Single Value in an Attribute

To specify a value in an attribute, set the `mod_op`, `mod_type`, and `mod_values` fields in an `LDAPMod` structure. (See the section "Specifying a New or Modified Attribute" for a brief overview of this structure.)

The following section of code sets up the structure for the `sn` attribute:

Code Example 8-2 Setting up an attribute structure

```
#include "ldap.h"
...
LDAPMod attribute1;
char *sn_values[] = { "Jensen", NULL };
...
attribute1.mod_op = 0;
attribute1.mod_type = "sn";
attribute1.mod_values = sn_values;
...
```

Note that because you are specifying an attribute for a new entry (rather than for an existing entry), you can set the `mod_op` field to `0`. (For an existing entry, the `mod_op` field identifies the type of change you are making to the entry.)

Specifying Multiple Values in an Attribute

If an attribute has more than one value, specify the values in the `mod_values` array. This example specifies two values for the `cn` attribute:

Code Example 8-3 Specifying multiple values in an attribute

```
#include "ldap.h"
...
LDAPMod attribute2, attribute3;
char *cn_values[] = { "Barbara Jensen", "Babs Jensen", NULL };
char *objectClass_values[] = { "top", "person",
  "organizationalPerson", "inetOrgPerson", NULL };
...
attribute2.mod_op = 0;
attribute2.mod_type = "cn";
attribute2.mod_values = cn_values;
attribute3.mod_op = 0;
attribute3.mod_type = "objectClass";
attribute3.mod_values = objectClass_values;
...
```

Specifying Binary Data as the Values of an Attribute

If the attribute contains binary data (rather than string values), specify the data in a `berval` structure:

```
struct berval {
  unsigned long bv_len;
  char *bv_val;
}
```

Table 8-2 berval field descriptions

bv_len	The length of the data.
bv_val	A pointer to the binary data.

After creating the berval structures for the binary data, you need to do the following:

- Add the berval structures to the mod_bvalues field in the LDAPMod structure.
- Use the bitwise OR operator (|) to combine the value of the mod_op field with LDAP_MOD_BVALUES. (When adding a new entry, you can just set the mod_op field to LDAP_MOD_BVALUES, since the mod_op field is 0 in this case.)

For example, suppose the file my_photo.jpg contains a JPEG photograph of Barbara Jensen. The following example sets the jpegPhoto attribute to the JPEG data of the photograph.

Code Example 8-4 Adding a value to an attribute

```
#include <stdio.h>
#include <sys/stat.h>
#include "ldap.h"
...
char *photo_data;
FILE *fp;
struct stat st;
LDAPMod attribute4;
struct berval photo_berval;
struct berval *jpegPhoto_values[2];
/* Get information about the JPEG file, including its size. */
if ( stat( "my_photo.jpg", &st ) != 0 ) {
  perror( "stat" );
  return( 1 );
}

/* Open the JPEG file and read it into memory. */
if ( ( fp = fopen( "my_photo.jpg", "rb" ) ) == NULL ) {
  perror( "fopen" );
  return( 1 );
}
if ( ( ( photo_data = ( char * )malloc( st.st_size ) ) == NULL ) ||
```

Code Example 8-4 Adding a value to an attribute

```
      ( fread ( photo_data, st.st_size, 1, fp ) != 1 ) ) {
  perror( photo_data ? "fread" : "malloc" );
  return( 1 );
}

fclose( fp );

attribute4.mod_op = LDAP_MOD_BVALUES;
attribute4.mod_type = "jpegPhoto";
photo_berval.bv_len = st.st_size;
photo_berval.bv_val = photo_data;
jpegPhoto_values[0] = &photo_berval;
jpegPhoto_values[1] = NULL;
attribute4.mod_values = jpegPhoto_values;
```

Specifying the Attributes in the Entry

After specifying the values of attributes in `LDAPMod` structures, you need to construct an array of these structures. (You will pass a pointer to this array as a parameter to the LDAP API function for creating a new entry.)

NOTE Make sure that you created `LDAPMod` structures for all required attributes in the new entry. For information on which attributes are required, see the documentation for your LDAP server. (For example, if you are using the iPlanet Directory Server, see the chapters on object classes and attributes in the *Directory Server Administrator's Guide*.)

The following section of code creates an array of `LDAPMod` structures:

Code Example 8-5 Adding an array of structures to an attribute

```
#include "ldap.h"
LDAPMod *list_of_mods[5]
LDAPMod attribute1, attribute2, attribute3, attribute4;
...
/* Code for filling the LDAPMod structures with values */
...
list_of_mods[0] = &attribute1;
list_of_mods[1] = &attribute2;
list_of_mods[2] = &attribute3;
list_of_mods[3] = &attribute4;
list_of_mods[4] = NULL;
...
```

Adding the Entry to the Directory

To add the entry to the directory, call one of the following functions:

- The synchronous `ldap_add_ext_s()` function (see "Performing a Synchronous Add Operation").
- The asynchronous `ldap_add_ext()` function (see "Performing an Asynchronous Add Operation").

(For more information about the difference between synchronous and asynchronous functions, see "Calling Synchronous and Asynchronous Functions.")

If you've allocated `LDAPMod` structures yourself, you should free the structures when you are done. Call the `ldap_mods_free()` function to free `LDAPMod` structures.

NOTE Before you add an entry, you should make sure that you've defined all required attributes for the entry type. For a listing of the required attributes for the different types of entries, see the documentation for your LDAP server. (For example, for the iPlanet Directory Server, see the chapters on object classes and attributes in the *Directory Server Administrator's Guide*.)

Performing a Synchronous Add Operation

If you want to wait for the results of the add operation to complete before continuing, call the synchronous `ldap_add_ext_s()` function. This function sends an LDAP add request to the server and blocks until the server sends the results of the operation back to your client.

The `ldap_add_ext_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred. See the documentation on the `ldap_add_ext_s()` function for a list of the possible result codes.

The following section of code uses the synchronous `ldap_add_ext_s()` function to add the user `William Jensen` to the directory.

Code Example 8-6 Synchronously adding an entry to the directory database

```
#include <stdio.h>
#include "ldap.h"
...
#define NEW_DN "uid=wbjensen,ou=People,o=airius.com"
...
LDAP         *ld;
```

Code Example 8-6 Synchronously adding an entry to the directory database

```
LDAPMod      **mods;
char         *matched_msg = NULL, *error_msg = NULL;
int         rc;
...
/* Perform the add operation. */
rc = ldap_add_ext_s( ld, NEW_DN, mods, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_add_ext_s: %s\n", ldap_err2string( rc ) );
  ldap_get_lderrno( ld, &matched_msg, &error_msg );
  if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
  }
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
} else {
  printf( "%s added successfully.\n", NEW_DN );
}
...
```

Performing an Asynchronous Add Operation

If you want to perform other work (in parallel) while waiting for the entry to be added, call the asynchronous `ldap_add_ext()` function. This function sends an LDAP add request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

The `ldap_add_ext()` function passes back a message ID identifying the add operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the add operation. The function passes back the results in an `LDAPMessage` structure.

You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful. For a list of possible result codes for an LDAP add operation, see the result code documentation for the `ldap_add_ext_s()` function.

The following section of code calls `ldap_add_ext()` to add the user `William Jensen` to the directory.

Code Example 8-7 Performing an asynchronous add operation

```
#include <stdio.h>
#include "ldap.h"
...
#define NEW_DN "uid=wbjensen,ou=People,o=airius.com"

...
LDAP           *ld;
LDAPMessage    *res;
LDAPControl    **serverctrls;
char           *matched_msg = NULL, *error_msg = NULL;
char           **referrals;
int            i, rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval  zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP add request. */
rc = ldap_add_ext( ld, NEW_DN, mods, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}
...
/* Poll the server for the results of the add operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {
  case -1:
    /* An error occurred. */
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  case 0:
    /* The timeout period specified by zerotime was exceeded,
      so call ldap_result() again and continue polling. */
    break;
  default:
    /* The function has retrieved the results of the add operation. */
    finished = 1;

    /* Parse the result to determine the result of
      the add operation. */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }
```

Code Example 8-7 Performing an asynchronous add operation

```
    /* Check the results of the LDAP add operation. */
    if ( rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
      if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
      }
    } else {
      printf( "%s added successfully.\n", NEW_DN );
    }
  }
}
...
```

Example: Adding an Entry to the Directory (Synchronous)

The following sample program calls the synchronous `ldap_add_ext_s()` function to add a new user to the directory.

Code Example 8-8 Synchronous add

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define NEW_DN "uid=wbjensen,ou=People,o=airius.com"
#define  NUM_MODS 5
int
main( int argc, char **argv )
{
  LDAP           *ld;
  LDAPMod        **mods;
  char           *matched_msg = NULL, *error_msg = NULL;
    int          i, rc;
  char *object_vals[] = { "top", "person", "organizationalPerson",
    "inetOrgPerson", NULL };

char *cn_vals[] = { "William B Jensen", "William Jensen", "Bill Jensen",
```

Code Example 8-8 Synchronous add

```
    NULL };
    char *sn_vals[] = { "Jensen", NULL };
    char *givenname_vals[] = { "William", "Bill", NULL };
    char *telephonenumber_vals[] = { "+1 415 555 1212", NULL };
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
    }
    /* Bind to the server as the Directory Manager. */
    rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
    }
    /* Construct the array of LDAPMod structures representing the attributes
     of the new entry. */
    mods = ( LDAPMod ** ) malloc(( NUM_MODS + 1 ) * sizeof( LDAPMod * ));
    if ( mods == NULL ) {
    fprintf( stderr, "Cannot allocate memory for mods array\n" );
    exit( 1 );
    }
    for ( i = 0; i < NUM_MODS; i++ ) {
    if (( mods[ i ] = ( LDAPMod * ) malloc( sizeof( LDAPMod ))) == NULL ) {
      fprintf( stderr, "Cannot allocate memory for mods element\n" );
      exit( 1 );
    }
    }
    mods[ 0 ]->mod_op = 0;
    mods[ 0 ]->mod_type = "objectclass";
    mods[ 0 ]->mod_values = object_vals;
    mods[ 1 ]->mod_op = 0;
    mods[ 1 ]->mod_type = "cn";
    mods[ 1 ]->mod_values = cn_vals;
    mods[ 2 ]->mod_op = 0;
    mods[ 2 ]->mod_type = "sn";
    mods[ 2 ]->mod_values = sn_vals;
    mods[ 3 ]->mod_op = 0;
    mods[ 3 ]->mod_type = "givenname";
    mods[ 3 ]->mod_values = givenname_vals;
    mods[ 4 ]->mod_op = 0;
    mods[ 4 ]->mod_type = "telephonenumber";
    mods[ 4 ]->mod_values = telephonenumber_vals;
    mods[ 5 ] = NULL;
    /* Perform the add operation. */
```

Code Example 8-8 Synchronous add

```
    rc = ldap_add_ext_s( ld, NEW_DN, mods, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_add_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL & *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    } else {
    printf( "%s added successfully.\n", NEW_DN );
  }
  ldap_unbind_s( ld );
  for ( i = 0; i < NUM_MODS; i++ ) {
    free( mods[ i ] );
  }
  free( mods );
  return 0;
}
```

Example: Adding an Entry to the Directory (Asynchronous)

The following sample program calls the asynchronous `ldap_add_ext()` function to add a new user to the directory.

Code Example 8-9 Asynchronous add

```
#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
void free_mods( LDAPMod **mods );
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define NEW_DN "uid=wbjensen,ou=People,o=airius.com"
#define  NUM_MODS 5
int
main( int argc, char **argv )
{
    LDAP      *ld;
```

Code Example 8-9 Asynchronous add

```
    LDAPMessage    *res;
    LDAPMod      **mods;
  LDAPControl    **serverctrls;
  char           *matched_msg = NULL, *error_msg = NULL;
  char           **referrals;
    int        i, rc, parse_rc, msgid, finished = 0;
    struct timeval  zerotime;
    char *object_vals[] = { "top", "person", "organizationalPerson",
"inetOrgPerson", NULL };
    char *cn_vals[] = { "William B Jensen", "William Jensen", "Bill Jensen",
NULL };
    char *sn_vals[] = { "Jensen", NULL };
    char *givenname_vals[] = { "William", "Bill", NULL };
    char *telephonenumber_vals[] = { "+1 415 555 1212", NULL };
    zerotime.tv_sec = zerotime.tv_usec = 0L;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
    }
    /* Bind to the server as the Directory Manager. */
    rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
    }
    /* Construct the array of LDAPMod structures representing the attributes
     of the new entry. */
    mods = ( LDAPMod ** ) malloc(( NUM_MODS + 1 ) * sizeof( LDAPMod * ));
    if ( mods == NULL ) {
    fprintf( stderr, "Cannot allocate memory for mods array\n" );
    exit( 1 );
    }
    for ( i = 0; i < NUM_MODS; i++ ) {
    if (( mods[ i ] = ( LDAPMod * ) malloc( sizeof( LDAPMod ))) == NULL ) {
      fprintf( stderr, "Cannot allocate memory for mods element\n" );
      exit( 1 );
    }
    }
    mods[ 0 ]->mod_op = 0;
    mods[ 0 ]->mod_type = "objectclass";
    mods[ 0 ]->mod_values = object_vals;
    mods[ 1 ]->mod_op = 0;
    mods[ 1 ]->mod_type = "cn";
```

Code Example 8-9 Asynchronous add

```
    mods[ 1 ]->mod_values = cn_vals;
    mods[ 2 ]->mod_op = 0;
    mods[ 2 ]->mod_type = "sn";
    mods[ 2 ]->mod_values = sn_vals;
    mods[ 3 ]->mod_op = 0;
    mods[ 3 ]->mod_type = "givenname";
    mods[ 3 ]->mod_values = givenname_vals;
    mods[ 4 ]->mod_op = 0;
    mods[ 4 ]->mod_type = "telephonenumber";
    mods[ 4 ]->mod_values = telephonenumber_vals;
    mods[ 5 ] = NULL;
    /* Send the LDAP add request. */
    rc = ldap_add_ext( ld, NEW_DN, mods, NULL, NULL, &msgid );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    free_mods( mods );
    return( 1 );
    }
    /* Poll the server for the results of the add operation. */
    while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
    case -1:
      /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      free_mods( mods );
      return( 1 );
    case 0:
      /* The timeout period specified by zerotime was exceeded.
         This means that the server has still not yet sent the
         results of the add operation back to your client.
         Break out of this switch statement, and continue calling
         ldap_result() to poll for results. */
      break;
    default:
      /* The function has retrieved the results of the add operation
         from the server. */
      finished = 1;
      /* Parse the results received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
&referrals, &serverctrls, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        free_mods( mods );
        return( 1 );
      }
```

Code Example 8-9 Asynchronous add

```
      /* Check the results of the LDAP add operation. */
      if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
          fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
        }
      } else {
        printf( "%s added successfully.\n"
          "Counted to %d while waiting for the add operation.\n",
          NEW_DN, global_counter );
      }
    }
    /* Do other work while waiting for the results of the add operation. */
    if ( !finished ) {
      do_other_work();
    }
  }
  ldap_unbind( ld );
  free_mods( mods );
  return 0;
}
/*
 * Free a mods array.
 */
void
free_mods( LDAPMod **mods )
{
    int i;
    for ( i = 0; i < NUM_MODS; i++ ) {
  free( mods[ i ] );
    }
    free( mods );
}

/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}
```

Modifying an Entry

To modify an entry:

1. Create an array of `LDAPMod` structures representing the changes that need to be made. Use the `LDAPMod` structure to specify a change to an attribute.
2. Call the `ldap_modify_ext()` or `ldap_modify_ext_s()`, passing in the array of `LDAPMod` structures and a distinguished name of the entry that you want modified.

After you are done, call the `ldap_mods_free()` function to free any `LDAPMod` structures you've allocated.

Specifying the Changes

This section describes the process of specifying changes to an entry.

- Replacing the Values of an Attribute
- Removing Values from an Attribute
- Adding Values to an Attribute
- Removing an Attribute
- Adding an Attribute
- Assembling the List of Changes

Replacing the Values of an Attribute

To replace all existing values of an attribute, create an `LDAPMod` structure with the following:

- Set the `mod_type` field to the attribute type that you want to change (for example, "`telephoneNumber`").
- Set the `mod_values` field to the new values of the attribute.
- Set the value of the `mod_op` field to `LDAP_MOD_REPLACE`.

If you want to specify binary data as berval structures (as opposed to string values), you need to do the following:

- Instead of using the `mod_values` field, use the `mod_bvalues` field.

- Use the bitwise OR operator (|) to combine the value LDAP_MOD_BVALUES with the value of the mod_op field.

Note the following:

- If you specify an attribute that does not exist in the entry, the attribute will be added to the entry.
- If you set a NULL value for the attribute (either by setting the mod_values field to NULL, or by setting the mod_bvalues field to NULL when the mod_op field contains LDAP_MOD_BVALUES), the attribute will be removed from the entry.

The following section of code specifies a change that replaces the values of the telephoneNumber attribute.

Code Example 8-10 Modifying attribute values

```
#include "ldap.h"
LDAPMod attribute1;
char *telephoneNumber_values[] = { "+1 650 555 1967", NULL };
attribute1.mod_type = "telephoneNumber";
attribute1.mod_op = LDAP_MOD_REPLACE;
attribute1.mod_values = telephoneNumber_values;
```

Removing Values from an Attribute

To remove values from an attribute, create an LDAPMod structure with the following:

- Set the mod_type field to the attribute type containing the values that you want to remove (for example, "facsimileTelephoneNumber").
- Set the mod_values field to the values that you want removed from the attribute.
- Set the value of the mod_op field to LDAP_MOD_DELETE.

If you want to specify binary data as berval structures (as opposed to string values), you need to do the following:

- Instead of using the mod_values field, use the mod_bvalues field.
- Use the bitwise OR operator (|) to combine the value LDAP_MOD_BVALUES with the value of the mod_op field.

Note the following:

- If you remove all values from the attribute, the attribute will be removed from the entry.
- If you set a `NULL` value for the attribute (either by setting the `mod_values` field to `NULL`, or by setting the `mod_bvalues` field to `NULL` when the `mod_op` field contains `LDAP_MOD_BVALUES`), the attribute will be removed from the entry.

The following example specifies the removal of one of the values of the `facsimileTelephoneNumber` attribute in the entry:

Code Example 8-11 Removing an attribute value

```
#include "ldap.h"
LDAPMod attribute2;
char *fax_values[] = { "+1 650 555 1967", NULL };
attribute2.mod_type = "facsimileTelephoneNumber";
attribute2.mod_op = LDAP_MOD_DELETE;
attribute2.mod_values = fax_values;
...
```

Adding Values to an Attribute

To add values to an attribute to an entry, create an `LDAPMod` structure with the following:

- Set the `mod_type` field to the attribute type that you want to add values to (for example, “audio”).
- Set the `mod_values` field to the new values of the attribute.
- Set the value of the `mod_op` field to `LDAP_MOD_ADD`.

If the attribute contains binary data (as opposed to string values), you need to do the following:

- Instead of using the `mod_values` field, use the `mod_bvalues` field. Make sure to put the values in `berval` structures.
- Use the bitwise OR operator (|) to combine the value `LDAP_MOD_BVALUES` with the value of the `mod_op` field.

Note that if the attribute does not already exist in the entry, the attribute will be added to the entry.

The following example adds values to the `audio` attribute to an entry:

Code Example 8-12 Adding values to an attribute

```
#include <stdio.h>
#include <sys/stat.h>
#include "ldap.h"
...
char *audio_data;
FILE *fp;
struct stat st;
LDAPMod attribute3;
struct berval audio_berval;
struct berval *audio_values[2];
...
/* Get information about the audio file, including its size. */
if ( stat( "my_sounds.au", &st ) != 0 ) {
  perror( "stat" );
  return( 1 );
}

/* Open the audio file and read it into memory. */
if ( ( fp = fopen( "my_sounds.au", "rb" ) ) == NULL ) {
  perror( "fopen" );
  return( 1 );
}

if ( ( ( audio_data = ( char * )malloc( st.st_size ) ) == NULL ) ||
      ( fread ( audio_data, st.st_size, 1, fp ) != 1 ) ) {
  perror( audio_data ? "fread" : "malloc" );
  return( 1 );
}

fclose( fp );
attribute3.mod_op = LDAP_MOD_ADD | LDAP_MOD_BVALUES;
attribute3.mod_type = "audio";
audio_berval.bv_len = st.st_size;
audio_berval.bv_val = audio_data;
audio_values[0] = &audio_berval;
audio_values[1] = NULL;
attribute3.mod_values = audio_values;
...
```

Removing an Attribute

You can remove an attribute from an entry in one of the following ways:

- Remove all values from the attribute.
- Set the `mod_op` field to `LDAP_MOD_REPLACE` or `LDAP_MOD_DELETE` and specify `NULL` for the `mod_values` field.

Adding an Attribute

If you add or replace values in an attribute that does not yet exist in the entry, the attribute will be added to the entry.

Assembling the List of Changes

After specifying the changes to attribute values in `LDAPMod` structures, you need to construct an array of these structures. (You will pass a pointer to this array as a parameter to the LDAP API function for modifying the entry.)

The following section of code creates an array of `LDAPMod` structures:

Code Example 8-13 Assembling a list of changes

```
#include "ldap.h"
...
LDAPMod *list_of_mods[4]
LDAPMod attribute1, attribute2, attribute3;
...
/* Code for filling the LDAPMod structures with values */
...
list_of_mods[0] = &attribute1;
list_of_mods[1] = &attribute2;
list_of_mods[2] = &attribute3;
list_of_mods[3] = NULL;
...
```

Modifying the Entry in the Directory

To modify the entry in the directory, call one of the following functions:

- The synchronous `ldap_modify_ext_s()` function (see “Performing a Synchronous Modify Operation”).
- The asynchronous `ldap_modify_ext()` function (see “Performing an Asynchronous Modify Operation”).

For more information about the difference between synchronous and asynchronous functions, see “Calling Synchronous and Asynchronous Functions.”

Performing a Synchronous Modify Operation

If you want to wait for the results of the modify operation to complete before continuing, call the synchronous `ldap_modify_ext_s()` function. This function sends an LDAP modify request to the server and blocks until the server sends the results of the operation back to your client.

The `ldap_modify_ext_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred. See the documentation on the `ldap_modify_ext_s()` function for a list of the possible result codes.

The following section of code uses the synchronous `ldap_add_ext_s()` function to modify the entry for the user `William Jensen` in the directory.

Code Example 8-14 Performing a synchronous modify operation

```
#include <stdio.h>
#include "ldap.h"

#define MODIFY_DN "uid=wbjensen,ou=People,o=airius.com"
...
LDAP      *ld;
LDAPMod   *mods[ 3 ];
char      *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform the modify operation. */
rc = ldap_modify_ext_s( ld, MODIFY_DN, mods, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_modify_ext_s: %s\n", ldap_err2string( rc ) );
  ldap_get_lderrno( ld, &matched_msg, &error_msg );
  if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
  }
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
} else {
  printf( "%s modified successfully.\n", MODIFY_DN );
}
ldap_unbind_s( ld );
...
```

Performing an Asynchronous Modify Operation

If you want to perform other work (in parallel) while waiting for the entry to be modified, call the asynchronous `ldap_modify_ext()` function. This function sends an LDAP modify request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent or it sends an LDAP result code if an error occurred.

The `ldap_modify_ext()` function passes back a message ID identifying the modify operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the modify operation. The function passes back the results in an `LDAPMessage` structure.

You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful. For a list of possible result codes for an LDAP modify operation, see the result code documentation for the `ldap_modify_ext_s()` function.

The following section of code calls `ldap_modify_ext()` to modify the entry for the user `William Jensen` in the directory.

Code Example 8-15 Performing an asynchronous modify operation

```
#include <stdio.h>
#include "ldap.h"
...
#define MODIFY_DN "uid=wbjensen,ou=People,o=airius.com"
...
LDAP      *ld;
LDAPMessage    *res;
LDAPMod      *mods[ 3 ];
LDAPControl    **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval  zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP modify request. */
rc = ldap_modify_ext( ld, MODIFY_DN, mods, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_modify_ext: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}
```

Code Example 8-15 Performing an asynchronous modify operation

```
/* Poll the server for the results of the modify operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {
  case -1:
    /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      return( 1 );
  case 0:
    /* The timeout period specified by zerotime was exceeded,
      so call ldap_result() again and continue polling. */
    break;
  default:
    /* The function has retrieved the results of the
      modify operation. */
    finished = 1;

    /* Parse the results received from the server. */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }

    /* Check the results of the LDAP modify operation. */
    if ( rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_modify_ext: %s\n",
        ldap_err2string( rc ) );
      if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
      }
    } else {
      printf( "%s modified successfully.\n", MODIFY_DN );
    }
  }
}
ldap_unbind( ld );
...
```

Example: Modifying an Entry in the Directory (Synchronous)

The following sample program calls the synchronous `ldap_modify_ext_s()` function to modify a user's entry in the directory. The program replaces the values of the mail attribute of the entry and adds a description attribute to the entry.

Code Example 8-16 Synchronous modify

```
#include <stdio.h>
#include <time.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define MODIFY_DN "uid=wbjensen,ou=People,o=airius.com"
int
main( int argc, char **argv )
{
  LDAP         *ld;
  LDAPMod      mod0, mod1;
  LDAPMod      *mods[ 3 ];
  char         *matched_msg = NULL, *error_msg = NULL;
  char         *vals0[ 2 ], *vals1[ 2 ];
  time_t       now;
  char         buf[ 128 ];
  int          rc;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind to the server as the Directory Manager. */
  rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Construct the array of LDAPMod structures representing the changes that you
want to make to attributes in the entry. */
/* Specify the first modification, which replaces all values of the
```

Code Example 8-16 Synchronous modify

```
mail attribute with the value "wbj@airius.com". */
  mod0.mod_op = LDAP_MOD_REPLACE;
  mod0.mod_type = "mail";
  vals0[0] = "wbj@airius.com";
  vals0[1] = NULL;
  mod0.mod_values = vals0;
/* Specify the second modification, which adds a value to the description
attribute. If this attribute does not yet exist, the attribute ia added to the
entry. */
  mod1.mod_op = LDAP_MOD_ADD;
  mod1.mod_type = "description";
  time( &now );
  sprintf( buf, "This entry was modified with the modattrs program on %s",
  ctime( &now ));
/* Get rid of \n which ctime put on the end of the time string */
  if ( buf[ strlen( buf ) - 1 ] == '\n' ) {
    buf[ strlen( buf ) - 1 ] = '\0';
  }
  vals1[ 0 ] = buf;
  vals1[ 1 ] = NULL;
  mod1.mod_values = vals1;
  mods[ 0 ] = &mod0;
  mods[ 1 ] = &mod1;
  mods[ 2 ] = NULL;
/* Perform the modify operation. */
  rc = ldap_modify_ext_s( ld, MODIFY_DN, mods, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_modify_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
  } else {
    printf( "%s modified successfully.\n", MODIFY_DN );
  }
  ldap_unbind_s( ld );
  return 0;
}
```

Example: Modifying an Entry in the Directory (Asynchronous)

The following sample program calls the asynchronous `ldap_modify_ext()` function to modify a user's entry in the directory. The program replaces the values of the mail attribute of the entry and adds a description attribute to the entry.

Code Example 8-17 Asynchronous modify

```
#include <stdio.h>
#include <time.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define MODIFY_DN "uid=wbjensen,ou=People,o=airius.com"
int
main( int argc, char **argv )
{
  LDAP           *ld;
  LDAPMessage    *res;
  LDAPMod        mod0, mod1;
  LDAPMod        *mods[ 3 ];
  LDAPControl    **serverctrls;
  char           *matched_msg = NULL, *error_msg = NULL;
  char           **referrals;
  char           *vals0[ 2 ], *vals1[ 2 ];
  time_t      now;
  char      buf[ 128 ];
  int        rc, parse_rc, msgid, finished = 0;
  struct timeval  zerotime;
  zerotime.tv_sec = zerotime.tv_usec = 0L;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind to the server as the Directory Manager. */
  rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
```

Code Example 8-17 Asynchronous modify

```
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Construct the array of LDAPMod structures representing the changes that you
want to make to attributes in the entry. */
/* Specify the first modification, which replaces all values of the
mail attribute with the value "wbj@airius.com". */
  mod0.mod_op = LDAP_MOD_REPLACE;
  mod0.mod_type = "mail";
  vals0[0] = "wbj@airius.com";
  vals0[1] = NULL;
  mod0.mod_values = vals0;
/* Specify the second modification, which adds a value to the description
attribute.  If this attribute does not yet exist, the attribute ia added to the
entry. */
  mod1.mod_op = LDAP_MOD_ADD;
  mod1.mod_type = "description";
  time( &now );
  sprintf( buf, "This entry was modified with the modattrs program on %s",
ctime( &now ));
/* Get rid of \n which ctime put on the end of the time string */
  if ( buf[ strlen( buf ) - 1 ] == '\n' ) {
    buf[ strlen( buf ) - 1 ] = '\0';
  }
  vals1[ 0 ] = buf;
  vals1[ 1 ] = NULL;
  mod1.mod_values = vals1;
  mods[ 0 ] = &mod0;
  mods[ 1 ] = &mod1;
  mods[ 2 ] = NULL;
/* Send the LDAP modify request. */
  rc = ldap_modify_ext( ld, MODIFY_DN, mods, NULL, NULL, &msgid );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_modify_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
/* Poll the server for the results of the modify operation. */
  while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
    case -1:
      /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      return( 1 );
    case 0:
      /* The timeout period specified by zerotime was exceeded.
         This means that the server has still not yet sent the
         results of the modify operation back to your client.
         Break out of this switch statement, and continue calling
         ldap_result() to poll for results. */
```

Code Example 8-17 Asynchronous modify

```
      break;
    default:
      /* The function has retrieved the results of the modify operation from
the server. */
      finished = 1;
      /* Parse the results received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
&referrals, &serverctrls, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      /* Check the results of the LDAP add operation. */
      if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_modify_ext: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
          fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
        }
      } else {
        printf( "%s modified successfully.\n"
          "Counted to %d while waiting for the modify operation.\n",
          MODIFY_DN, global_counter );
      }
    }
/* Do other work while waiting for the results of the modify operation. */
    if ( !finished ) {
      do_other_work();
    }
  }
  ldap_unbind( ld );
  return 0;
}

/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
  global_counter++;
}
```

Deleting an Entry

To remove an entry from the directory, call one of the following functions:

- The synchronous `ldap_delete_ext_s()` function (see “Performing a Synchronous Delete Operation”).
- The asynchronous `ldap_delete_ext()` function (see “Performing an Asynchronous Delete Operation”).

For more information about the difference between synchronous and asynchronous functions, see “Calling Synchronous and Asynchronous Functions.”

Performing a Synchronous Delete Operation

If you want to wait for the results of the delete operation to complete before continuing, call the synchronous `ldap_delete_ext_s()` function. This function sends an LDAP delete request to the server and blocks until the server sends the results of the operation back to your client.

The `ldap_delete_ext_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred. See the documentation on the `ldap_delete_ext_s()` function for a list of the possible result codes.

The following section of code uses the synchronous `ldap_delete_ext_s()` function to remove the entry for the user `William Jensen` from the directory.

Code Example 8-18 Synchronous deletion

```
#include <stdio.h>
#include "ldap.h"
...
#define DELETE_DN "uid=wjensen,ou=People,o=airius.com"
...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int      rc;
...
/* Perform the delete operation. */
rc = ldap_delete_ext_s( ld, DELETE_DN, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_delete_ext_s: %s\n", ldap_err2string( rc ) );
  ldap_get_lderrno( ld, &matched_msg, &error_msg );
  if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
  }
```

Code Example 8-18 Synchronous deletion

```
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
} else {
  printf( "%s deleted successfully.\n", DELETE_DN );
}
ldap_unbind_s( ld );
...
```

Performing an Asynchronous Delete Operation

If you want to perform other work (in parallel) while waiting for the entry to be deleted, call the asynchronous `ldap_delete_ext()` function. This function sends an LDAP delete request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

The `ldap_delete_ext()` function passes back a message ID identifying the delete operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the delete operation. The function passes back the results in an `LDAPMessage` structure.

You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful. For a list of possible result codes for an LDAP delete operation, see the result code documentation for the `ldap_delete_ext_s()` function.

The following section of code calls `ldap_delete_ext()` to remove the user `William Jensen` from the directory.

Code Example 8-19 Asynchronous deletion

```
#include <stdio.h>
#include "ldap.h"
...
#define DELETE_DN "uid=wjensen,ou=People,o=airius.com"
...
LDAP      *ld;
LDAPMessage    *res;
LDAPControl    **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
```

Code Example 8-19 Asynchronous deletion

```
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval  zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP delete request. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}

/* Poll the server for the results of the delete operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {
  case -1:
    /* An error occurred. */
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  case 0:
    /* The timeout period specified by zerotime was exceeded,
      so call ldap_result() again and continue polling. */
      break;
  default:
    /* The function has retrieved the results of the
      delete operation. */
      finished = 1;

    /* Parse the results received from the server. */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }
    /* Check the results of the LDAP delete operation. */
    if ( rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_delete_ext: %s\n",
        ldap_err2string( rc ) );
      if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
```

Code Example 8-19 Asynchronous deletion

```
      }
    } else {
      printf( "%s deleted successfully.\n", DELETE_DN );
    }
  }
}
ldap_unbind( ld );
...
```

Example: Deleting an Entry from the Directory (Synchronous)

The following sample program calls the synchronous `ldap_delete_ext_s()` function to delete a user's entry from the directory.

Code Example 8-20 Synchronous entry deletion

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define DELETE_DN "uid=wjensen,ou=People,o=airius.com"
int
main( int argc, char **argv )
{
  LDAP      *ld;
  char      *matched_msg = NULL, *error_msg = NULL;
  int        rc;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind to the server as the Directory Manager. */
  rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
```

Code Example 8-20 Synchronous entry deletion

```
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Perform the delete operation. */
  rc = ldap_delete_ext_s( ld, DELETE_DN, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
  } else {
    printf( "%s deleted successfully.\n", DELETE_DN );
  }
  ldap_unbind_s( ld );
  return 0;
}
```

Example: Deleting an Entry from the Directory (Asynchronous)

The following sample program calls the asynchronous `ldap_delete_ext()` function to delete a user's entry from the directory.

Code Example 8-21 Asynchronous entry deletion

```
#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define DELETE_DN "uid=wjensen,ou=People,o=airius.com"
int
main( int argc, char **argv )
{
  LDAP      *ld;
```

Code Example 8-21 Asynchronous entry deletion

```
  LDAPMessage    *res;
  LDAPControl    **serverctrls;
  char      *matched_msg = NULL, *error_msg = NULL;
  char      **referrals;
  int        rc, parse_rc, msgid, finished = 0;
  struct timeval  zerotime;
  zerotime.tv_sec = zerotime.tv_usec = 0L;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind to the server as the Directory Manager. */
  rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Send the LDAP delete request. */
  rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
/* Poll the server for the results of the delete operation. */
  while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
    case -1:
      /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      return( 1 );
    case 0:
      /* The timeout period specified by zerotime was exceeded.
         This means that the server has still not yet sent the
         results of the delete operation back to your client.
         Break out of this switch statement, and continue calling
         ldap_result() to poll for results. */
      break;
    default:
  /* The function has retrieved the results of the delete operation from the
server. */
```

Code Example 8-21 Asynchronous entry deletion

```
      finished = 1;
      /* Parse the results received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
&referrals, &serverctrls, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      /* Check the results of the LDAP delete operation. */
      if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
        }
      } else {
        printf( "%s deleted successfully.\n"
          "Counted to %d while waiting for the delete operation.\n",
          DELETE_DN, global_counter );
      }
    }
    /* Do other work while waiting for the results of the delete operation. */
    if ( !finished ) {
      do_other_work();
    }
  }
  ldap_unbind( ld );
  return 0;
}
/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}
```

Changing the DN of an Entry

To change the distinguished name (DN) of an entry, call one of the following functions:

- The synchronous `ldap_rename_s()` function (see “Performing a Synchronous Renaming Operation”).
- The asynchronous `ldap_rename()` function (see “Performing an Asynchronous Renaming Operation”).

For more information about the difference between synchronous and asynchronous functions, see “Calling Synchronous and Asynchronous Functions.”

For both functions, you can choose to delete the attribute that represents the old relative distinguished name (RDN) (see“Deleting the Attribute from the Old RDN” for details). You can also change the location of the entry in the directory tree (see “Changing the Location of the Entry” for details).

Performing a Synchronous Renaming Operation

If you want to wait for the results of the modify DN operation to complete before continuing, call the synchronous `ldap_rename_s()` function. This function sends an LDAP modify DN request to the server and blocks until the server sends the results of the operation back to your client.

The `ldap_rename_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred. See the documentation on the `ldap_rename_s()` function for a list of the possible result codes.

The following section of code uses the synchronous `ldap_rename_s()` function to change the RDN of the entry for the user `William Jensen` in the directory.

Code Example 8-22 Performing a synchronous modification

```
#include <stdio.h>
#include "ldap.h"
...
#define OLD_DN "uid=wbjensen,ou=People,o=airius.com"
#define NEW_RDN "uid=wjensen"
...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform the modify DN operation. */
```

Code Example 8-22 Performing a synchronous modification

```
rc = ldap_rename_s( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_rename_s: %s\n", ldap_err2string( rc ) );
  ldap_get_lderrno( ld, &matched_msg, &error_msg );
  if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
  }
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
} else {
  printf( "%s renamed successfully.\n", OLD_DN );
}
ldap_unbind_s( ld );
...
```

Performing an Asynchronous Renaming Operation

If you want to perform other work (in parallel) while waiting for the entry to be renamed, call the asynchronous `ldap_rename()` function. This function sends an LDAP modify DN request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

The `ldap_rename()` function passes back a message ID identifying the modify DN operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the modify DN operation. The function passes back the results in an `LDAPMessage` structure.

You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful. For a list of possible result codes for an LDAP rename operation, see the result code documentation for the `ldap_rename_s()` function.

The following section of code calls `ldap_rename()` to change the RDN of the user `William Jensen` in the directory.

Code Example 8-23 Performing an asynchronous modification

```
#include <stdio.h>
#include "ldap.h"
...
#define OLD_DN "uid=wbjensen,ou=People,o=airius.com"
#define NEW_RDN "uid=wjensen"
...
LDAP      *ld;
LDAPMessage    *res;
LDAPControl    **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval  zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP modify DN request. */
rc = ldap_rename( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
  fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}

/* Poll the server for the results of the modify DN operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {
  case -1:
    /* An error occurred. */
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  case 0:
    /* The timeout period specified by zerotime was exceeded,
      so call ldap_result() again and continue polling. */
    break;
  default:
    /* The function has retrieved the results of the
      modify DN operation. */
    finished = 1;

    /* Parse the results received from the server. */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }
```

Code Example 8-23 Performing an asynchronous modification

```
    /* Check the results of the LDAP modify DN operation. */
    if ( rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
      if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
      }
    } else {
      printf( "%s renamed successfully.\n", OLD_DN );
    }
  }
}
ldap_unbind( ld );
...
```

Deleting the Attribute from the Old RDN

Both `ldap_rename()` and `ldap_rename_s()` have 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 `cn` attribute:

```
cn: Barbara Jensen
cn: Babs Jensen
```

The following function call adds "`Barbie Jensen`" to this list of values and removes the "`Barbara Jensen`" value:

```
ldap_modrdn2( "cn=Barbara Jensen", "cn=Barbie Jensen", 1 );
```

The resulting entry has the following values:

```
cn: Barbie Jensen
cn: Babs Jensen
```

If instead a `0` is passed for the `deleteoldrdn` parameter:

```
ldap_modrdn2( "cn=Barbara Jensen", "cn=Barbie Jensen", 0 );
```

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

```
cn: Barbie Jensen
cn: Babs Jensen
cn: Barbara Jensen
```

Changing the Location of the Entry

Both `ldap_rename()` and `ldap_rename_s()` have a `newparent` parameter that allows you to specify a new location for the entry in the directory tree.

For example, if you pass "`ou=Contractors, o=airius.com`" as the `newparent` parameter when renaming the entry "`uid=bjensen,ou=People,o=airius.com`", the entry is moved under "`ou=Contractors, o=airius.com`" and the new DN for the entry is "`uid=bjensen,ou=Contractors, o=airius.com`".

Note that not all LDAP servers support this feature. At this point in time, the Netscape Directory Server 4.*x* does not support this feature. If you specify this argument, the Netscape Directory Server will send back the LDAP result code `LDAP_UNWILLING_TO_PERFORM` with the error message "server does not support moving of entries."

Example: Renaming an Entry in the Directory (Synchronous)

The following sample program calls the synchronous `ldap_rename_s()` function to change the RDN of a user's entry in the directory.

Code Example 8-24 Synchronous renaming of an entry

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define OLD_DN "uid=wbjensen,ou=People,o=airius.com"
#define NEW_RDN "uid=wjensen"
int
main( int argc, char **argv )
{
  LDAP      *ld;
  char      *matched_msg = NULL, *error_msg = NULL;
  int        rc;
/* Get a handle to an LDAP connection. */
```

Code Example 8-24 Synchronous renaming of an entry

```
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind to the server as the Directory Manager. */
  rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Perform the modify DN operation. */
  rc = ldap_rename_s( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_rename_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
  } else {
    printf( "%s renamed successfully.\n", OLD_DN );
  }
  ldap_unbind_s( ld );
  return 0;
}
```

Example: Renaming an Entry in the Directory (Asynchronous)

The following sample program calls the asynchronous `ldap_rename()` function to change the RDN of a user's entry in the directory.

Code Example 8-25 Asynchronous renaming of an entry

```
#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "dougy4444"
#define OLD_DN "uid=wbjensen,ou=People,o=airius.com"
#define NEW_RDN "uid=wjensen"
int
main( int argc, char **argv )
{
  LDAP      *ld;
  LDAPMessage    *res;
  LDAPControl    **serverctrls;
  char      *matched_msg = NULL, *error_msg = NULL;
  char      **referrals;
  int        rc, parse_rc, msgid, finished = 0; struct timeval  zerotime;
  zerotime.tv_sec = zerotime.tv_usec = 0L;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind to the server as the Directory Manager. */
  rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Send the LDAP modify DN request. */
  rc = ldap_rename( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL, &msgid );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
/* Poll the server for the results of the modify DN operation. */
  while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
    case -1:
```

Code Example 8-25 Asynchronous renaming of an entry

```
      /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      return( 1 );
    case 0:
      /* The timeout period specified by zerotime was exceeded.
         This means that the server has still not yet sent the
         results of the modify DN operation back to your client.
         Break out of this switch statement, and continue calling
         ldap_result() to poll for results. */
      break;
    default:
  /* The function has retrieved the results of the modify DN operation
         from the server. */
      finished = 1;
      /* Parse the results received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
&referrals, &serverctrls, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      /* Check the results of the LDAP modify DN operation. */
      if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
          fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
        }
      } else {
        printf( "%s renamed successfully.\n"
          "Counted to %d while waiting for the modify DN operation.\n",
          OLD_DN, global_counter );
      }
    }
    /* Do other work while waiting for the results of the modify DN operation.
*/
    if ( !finished ) {
      do_other_work();
    }
  }
  ldap_unbind( ld );
  return 0;
}
```

Code Example 8-25 Asynchronous renaming of an entry

```
/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}
```

Chapter 9

Comparing Values in Entries

This chapter explains how to use the API functions to compare data in entries in the directory. The LDAP API includes functions that you can call to compare the value of an attribute in an entry against a specified value.

The chapter includes the following sections:

- Comparing the Value of an Attribute
- Performing a Synchronous Comparison Operation
- Performing an Asynchronous Comparison Operation

Comparing the Value of an Attribute

To determine if an attribute contains a certain value, call one of the following functions:

- The synchronous `ldap_compare_ext_s()` function (see “Performing a Synchronous Comparison Operation”).
- The asynchronous `ldap_compare_ext()` function (see “Performing an Asynchronous Comparison Operation”).

For more information about the difference between synchronous and asynchronous functions, see “Calling Synchronous and Asynchronous Functions.”

Note that both of these functions compare values that are in `berval` structures; this allows you to compare binary values. If you just want to compare string values, you can call the `ldap_compare()` or `ldap_compare_s()` function.

Performing a Synchronous Comparison Operation

If you want to wait for the results of the compare operation to complete before continuing, call the synchronous `ldap_compare_ext_s()` function to compare berval structures or wait for the results of the compare operation to complete before continuing, call the synchronous `ldap_compare_s()` function to compare string values. These functions send an LDAP compare request to the server and block until the server sends the results of the operation back to your client.

Both the `ldap_compare_ext_s()` function and the `ldap_compare_s()` function return one of the following values after the LDAP compare operation completes:

- `LDAP_COMPARE_TRUE` indicates that the attribute contains the specified value.
- `LDAP_COMPARE_FALSE` indicates that the attribute does not contain the specified value.
- An error code indicates that a problem has occurred during the operation.

See the documentation on the `ldap_compare_ext_s()` function for a list of the possible result codes.

The following section of code uses the synchronous `ldap_compare_s()` function to determine if an entry has the value "`bjensen2airius.com`" in the mail attribute.

Code Example 9-1 Applying the ldap_compare_s() function

```
#include <stdio.h>
#include "ldap.h"
...
#define COMPARE_DN "uid=bjensen,ou=People,o=airius.com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@airius.com"
...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int       rc;
...
/* Perform the compare operation. */
rc = ldap_compare_s( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
switch( rc ) {
case LDAP_COMPARE_TRUE:
  printf( "%s has the value %s in the %s attribute.\n", COMPARE_DN,
    COMPARE_VALUE, COMPARE_ATTR );
  break;
case LDAP_COMPARE_FALSE:
  printf( "%s does not have the value %s in the %s attribute.\n",
    COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
```

Code Example 9-1 Applying the ldap_compare_s() function

```
  break;
default:
  fprintf( stderr, "ldap_compare_s: %s\n", ldap_err2string( rc ) );
  ldap_get_lderrno( ld, &matched_msg, &error_msg );
  if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
  }
  if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
      "Part of the DN that matches an existing entry: %s\n",
      matched_msg );
  }
  break;
}
ldap_unbind_s( ld );
...
```

Performing an Asynchronous Comparison Operation

If you want to perform other work (in parallel) while waiting for the comparison to complete, call the asynchronous `ldap_compare_ext()` function to compare values in berval structures or the asynchronous `ldap_compare()` function to compare string values. These functions send an LDAP compare request to the server and return an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

Both the `ldap_compare_ext()` function and the `ldap_compare()` function pass back a message ID identifying the compare operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the compare operation. The function passes back the results in an `LDAPMessage` structure.

You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful. The result code should be one of the following:

- `LDAP_COMPARE_TRUE` indicates that the attribute contains the specified value.
- `LDAP_COMPARE_FALSE` indicates that the attribute does not contain the specified value.

- An error code indicates that a problem occurred during the operation. (For a list of possible result codes for an LDAP compare operation, see the result code documentation for the `ldap_compare_ext_s()` function.)

The following section of code calls `ldap_compare()` to determine if an entry has the value "`bjensen@airius.com`" in the mail attribute.

Code Example 9-2 Applying the ldap_compare() function

```
#include <stdio.h>
#include "ldap.h"
...
#define COMPARE_DN "uid=bjensen,ou=People,o=airius.com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@airius.com"
...
LDAP      *ld;
LDAPMessage    *res;
LDAPControl    **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;
struct timeval  zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP compare request. */
msgid = ldap_compare( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
if ( msgid < 0 ) {
  fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
  ldap_unbind( ld );
  return( 1 );
}
/* Poll the server for the results of the LDAP compare operation. */
while ( !finished ) {
  rc = ldap_result( ld, msgid, 0, &zerotime, &res );
  switch ( rc ) {

  case -1:
    /* An error occurred. */
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );

  case 0:
      /* The timeout period specified by zerotime was exceeded, so
         call ldap_result() again and continue to poll for the
        results. */
    break;

  default:
    /* The client has received the results of the
       LDAP compare operation from the server. */
    finished = 1;
```

Code Example 9-2 Applying the ldap_compare() function

```
    /* Parse the results received from the server.*/
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
      &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
      fprintf( stderr, "ldap_parse_result: %s\n",
        ldap_err2string( parse_rc ) );
      ldap_unbind( ld );
      return( 1 );
    }

    /* Check the results of the LDAP compare operation. */
    switch ( rc ) {
    case LDAP_COMPARE_TRUE:
      printf( "%s has the value %s in the %s attribute.\n",
        COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
      break;
    case LDAP_COMPARE_FALSE:
      printf( "%s does not have the value %s in the %s attribute.\n",
        COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
      break;
    default:
      fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
      if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
      }
      if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
          "Part of the DN that matches an existing entry: %s\n",
          matched_msg );
      }
      break;
    }
  }
}
...
```

Example: Comparing a Value in an Entry (Synchronous)

The following sample program calls the synchronous `ldap_compare_s()` function to determine if a user's entry has the value "`bjensen@airius.com`" in the mail attribute.

Code Example 9-3 Comparing entry values using ldap_compare_s()

```
#include <stdio.h>
#include "ldap.h"

/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define COMPARE_DN "uid=bjensen,ou=People,o=airius.com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@airius.com"
int
main( int argc, char **argv )
{
  LDAP      *ld;
  char      *matched_msg = NULL, *error_msg = NULL;
  int        rc;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind anonymously to the server. */
  rc = ldap_simple_bind_s( ld, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
/* Perform the compare operation. */
  rc = ldap_compare_s( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
  switch( rc ) {
  case LDAP_COMPARE_TRUE:
    printf( "%s has the value %s in the %s attribute.\n", COMPARE_DN,
COMPARE_VALUE, COMPARE_ATTR );
    break;
  case LDAP_COMPARE_FALSE:
    printf( "%s does not have the value %s in the %s attribute.\n",
COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
    break;
  default:
    fprintf( stderr, "ldap_compare_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
```

Code Example 9-3 Comparing entry values using ldap_compare_s()

```
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
      fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
    }
    break;
  }
  ldap_unbind_s( ld );
  return 0;
}
```

Example: Comparing a Value in an Entry (Asynchronous)

The following sample program calls the asynchronous `ldap_compare()` function to determine if a user's entry has the value "`bjensen@airius.com`" in the mail attribute.

Code Example 9-4 Comparing entry values using ldap_compare()

```
#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define COMPARE_DN "uid=bjensen,ou=People,o=airius.com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@airius.com"
int
main( int argc, char **argv )
{
  LDAP      *ld;
  LDAPMessage    *res;
  LDAPControl    **serverctrls;
  char      *matched_msg = NULL, *error_msg = NULL;
  char      **referrals;
  int        rc, parse_rc, msgid, finished = 0;
  struct timeval  zerotime;
  zerotime.tv_sec = zerotime.tv_usec = 0L;
/* Get a handle to an LDAP connection. */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
/* Bind anonymously to the server. */
```

Code Example 9-4 Comparing entry values using ldap_compare()

```
  rc = ldap_simple_bind_s( ld, NULL, NULL );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, NULL, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
      fprintf( stderr, "%s\n", error_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
  }
  /* Send the LDAP compare request. */
  msgid = ldap_compare( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
  if ( msgid < 0 ) {
    fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
/* Poll the server for the results of the LDAP compare operation. */
  while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
    case -1:
      /* An error occurred. */
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      ldap_unbind( ld );
      return( 1 );
    case 0:
      /* The timeout period specified by zerotime was exceeded.
         This means that your client has not yet received the
         results of the LDAP compare operation.
         Break out of this switch statement, and continue calling
         ldap_result() to poll for the results. */
      break;
    default:
      /* The client has received the results of the
         LDAP compare operation from the server. */
      finished = 1;
      /* Parse the results received from the server. Note the last
         argument is a non-zero value, which indicates that the
         LDAPMessage structure will be freed when done.  (No need
         to call ldap_msgfree().) */
      parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
&referrals, &serverctrls, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc
) );
        ldap_unbind( ld );
        return( 1 );
      }
      /* Check the results of the LDAP compare operation. */
      switch ( rc ) {
      case LDAP_COMPARE_TRUE:
        printf( "%s has the value %s in the %s attribute.\n"
```

Code Example 9-4 Comparing entry values using ldap_compare()

```
          "Counted to %d while waiting for the compare operation.\n",
          COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR, global_counter );
        break;
      case LDAP_COMPARE_FALSE:
        printf( "%s does not have the value %s in the %s attribute.\n"
          "Counted to %d while waiting for the compare operation.\n",
          COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR, global_counter );
        break;
      default:
        fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
          fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
          fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
        }
        break;
      }
    }
    /* Do other work while waiting for the results of the compare operation. */
    if ( !finished ) {
      do_other_work();
    }
  }
  ldap_unbind( ld );
  return 0;
}
/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}
```

Chapter 10

Working with LDAP URLs

This chapter explains how you can call functions to parse an LDAP URL into its components and to process a search request specified by an LDAP URL.

The chapter contains the following sections:

- Understanding LDAP URLs
- Examples of LDAP URLs
- Determining If a URL is an LDAP URL
- Getting the Components of an LDAP URL
- Freeing 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 sent to an LDAP server.

LDAP URLs have the following syntax:

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

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.iPlanet.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.
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.iplanet.com/o=airius.com??sub?(sn=Jensen)
```

Note that the two consecutive question marks ("??") indicates 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.iplanet.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.iplanet.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.iplanet.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.iplanet.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.iplanet.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.

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

Determining If a URL is an LDAP URL

To determine whether a URL is an LDAP URL, call the `ldap_is_ldap_url()` function. This function returns a nonzero value if the URL is an LDAP URL. If the URL is not an LDAP URL, the function returns 0.

The following section of code determines if a URL is an LDAP URL.

Code Example 10-1 Parsing an LDAP URL

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url = "ldap://ldap.ipalnet.com/o=airius.com";
...
if ( ldap_is_ldap_url( my_url ) != 0 ) {
  printf( "%s is an LDAP URL.\n", my_url );
} else {
  printf( "%s is not an LDAP URL.\n", my_url );
}
...
```

To verify that a URL complies with the LDAP URL syntax, you should call the `ldap_url_parse()` function (see “Getting the Components of an LDAP URL”).

Getting the Components of an LDAP URL

To get the individual components of the URL, call the `ldap_url_parse()` function. This function returns the LDAP URL components in an `LDAPURLDesc` structure, which is shown here:

```
typedef struct ldap_url_desc {
  char *lud_host;
  int lud_port;
  char *lud_dn;
  char **lud_attrs;
  int lud_scope;
  char *lud_filter;
  unsigned long lud_options;
} LDAPURLDesc;
```

Here is a list of the field descriptions:

Table 10-2 ldap_url_desc field descriptions

`lud_host`	The name of the host in the URL.
`lud_port`	The number of the port in the URL.
`lud_dn`	The distinguished name in the URL.
`lud_attrs`	A pointer to a `NULL`-terminated array of the attributes specified in the URL.
`lud_scope`	The scope of the search specified in the URL. This field can have the following values: • `LDAP_SCOPE_BASE` specifies a search of the base entry. • `LDAP_SCOPE_ONELEVEL` specifies a search of all entries one level under the base entry (not including the base entry). • `LDAP_SCOPE_SUBTREE` specifies a search of all entries at all levels under the base entry (including the base entry).
`lud_filter`	Search filter included in the URL.
`lud_options`	Options (if `LDAP_URL_OPT_SECURE`, indicates that the protocol is `ldaps://` instead of `ldap://`).

The following section of code parses an LDAP URL and prints out each component of the URL.

Code Example 10-2 Parsing an LDAP URL

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url =
"ldap://ldap.iplanet.com:5000/o=airius.com?cn,mail,telephoneNumber?sub?
(sn=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
  switch( res ){
    case LDAP_URL_ERR_NOTLDAP:
      printf( "URL does not begin with \"ldap://\"\n" );
      break;
    case LDAP_URL_ERR_NODN:
      printf( "URL missing trailing slash after host or port\n" );
      break;
    case LDAP_URL_ERR_BADSCOPE:
      printf( "URL contains an invalid scope\n" );
      break;
    case LDAP_URL_ERR_MEM:
      printf( "Not enough memory\n" );
      break;
    default:
      printf( "Unknown error\n" );
  }
  return( 1 );
}
printf( "Components of the URL:\n" );
printf( "Host name: %s\n", ludpp->lud_host );
printf( "Port number: %d\n", ludpp->lud_port );
if ( ludpp->lud_dn != NULL ) {
  printf( "Base entry: %s\n", ludpp->lud_dn );
} else {
  printf( "Base entry: Root DN\n" );
}
if ( ludpp->lud_attrs != NULL ) {
  printf( "Attributes returned: \n" );
  for ( i=0; ludpp->lud_attrs[i] != NULL; i++ ) {
    printf( "\t%s\n", ludpp->lud_attrs[i] );
  }
} else {
  printf( "No attributes returned.\n" );
}
printf( "Scope of the search: " );
switch( ludpp->lud_scope ) {
  case LDAP_SCOPE_BASE:
    printf( "base\n" );
    break;
  case LDAP_SCOPE_ONELEVEL:
```

Code Example 10-2 Parsing an LDAP URL

```
    printf( "one\n" );
    break;
  case LDAP_SCOPE_SUBTREE:
    printf( "sub\n" );
    break;
  default:
    printf( "Unknown scope\n" );
}
printf( "Filter: %s\n", ludpp->lud_filter );
...
```

The code prints out the following results:

```
Components of the URL:
Host name: ldap.netscape.com
Port number: 5000
Base entry: o=airius.com
Attributes returned:
  cn
  mail
  telephoneNumber
Scope of the search: sub
Filter: (sn=Jensen)
```

Freeing the Components of an LDAP URL

When you have finished working with the components of an LDAP URL, you should free the `LDAPURLDesc` structure from memory by calling the `ldap_free_urldesc()` function.

The following section of code parses an LDAP URL and then frees the `LDAPURLDesc` structure from memory after verifying that the LDAP URL is valid.

Code Example 10-3 Using ldap_free_urldesc() to free the components of an LDAP URL

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url = "ldap://ldap.netscape.com:5000/o=airius.com?cn,mail, \
  telephoneNumber?sub?(sn=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
```

Code Example 10-3 Using ldap_free_urldesc() to free the components of an LDAP URL

```
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
  switch( res ){
    case LDAP_URL_ERR_NOTLDAP:
      printf( "URL does not begin with \"ldap://\"\n" );
      break;
    case LDAP_URL_ERR_NODN:
      printf( "URL does not contain a distinguished name\n" );
      break;
    case LDAP_URL_ERR_BADSCOPE:
      printf( "URL contains an invalid scope\n" );
      break;
    case LDAP_URL_ERR_MEM:
      printf( "Not enough memory\n" );
      break;
    default:
      printf( "Unknown error\n" );
  }
  return( 1 );
}
printf( "URL is a valid LDAP URL\n" );
ldap_free_urldesc( ludpp );
...
```

Processing an LDAP URL

To process an LDAP URL search request, call the `ldap_url_search_s()`, `ldap_url_search_st()`, or `ldap_url_search()` function.

- `ldap_url_search_s()` is a synchronous function that completes the search operation before returning. Call this function if you need to wait for the operation to finish before continuing.

  The `ldap_url_search_s()` function returns `LDAP_SUCCESS` if the operation completed successfully. If an error occurred, the function returns an error code. (See “Result Codes” for a complete listing of error codes.)

- `ldap_url_search_st()` is a synchronous function that allows a certain amount of time for the completion of the search operation. Call this function if you need to wait for the operation to complete and if you want to set a timeout period for the operation.

- `ldap_url_search()` is an asynchronous function that initiates the search operation but does not wait for the operation to complete. Call this function if you want to perform other work (in parallel) while waiting for the operation to complete.

The `ldap_url_search()` function returns a message ID identifying the search operation. To determine whether the operation is completed or still in progress, call the `ldap_result()` function.

After the operation is completed, call the `ldap_result2error()` function to determine whether the operation was successful. If the operation completed successfully, the `ldap_result2error()` function returns `LDAP_SUCCESS`. If an error occurred, the function returns an error code. See Chapter 19, “Result Codes” for a complete listing.

For more information about the difference between synchronous and asynchronous functions, see “Calling Synchronous and Asynchronous Functions.”

The following example processes a search request from an LDAP URL.

Code Example 10-4 Processing an LDAP URL search request

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_url = "ldap://ldap.netscape.com/o=airius.com?cn,mail, \
  telephoneNumber?sub?(sn=Jensen)";
/* Process the search request in the URL. */
if ( ldap_url_search_s( ld, my_url, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_url_search_s" );
  return( 1 );
}
```

Part 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.

The chapter includes the following sections:

- Understanding DSEs
- Getting the Root DSE
- Determining If the Server Supports LDAPv3
- 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 LDAPv3 protocol. LDAPv2servers 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 LDAPv3 extended operations supported by this server (see Chapter 15, “Working with Extended Operations” for details).

- The LDAPv3 controls supported by this server (see Chapter 14, “Working with LDAP Controls” for details).
- The SASL mechanisms supported by this server (see Chapter 13, “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 LDAPv3 extended operations supported by this server. If this attribute is not in the root DSE, the server does not support any extended operations.
`supportedControl`	The values of this attribute are the object identifiers (OIDs) of the LDAPv3 controls supported by this server. If this attribute is not in the root DSE, the server does not support any LDAPv3 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.

Table 11-1 Information available in the root DSE

Attribute Name	Description of Values
`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. Initialize an LDAP session by calling the `ldap_init()` function.
2. Turn off automatic referral handling by calling the `ldap_set_option()` function and setting the `LDAP_OPT_REFERRALS` option to `LDAP_OPT_OFF`.
3. 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 API functions to search the directory, see Chapter 6, "Searching the Directory."
4. Check the results of the search.

   If the server returns a result code such as `LDAP_OPERATIONS_ERROR`, `LDAP_PROTOCOL_ERROR`, `LDAP_REFERRAL`, or `LDAP_NO_SUCH_OBJECT` result code, the LDAP server probably does not support LDAPv3.

The following function gets the root DSE for a server and prints out its attributes. The function assumes that you are passing in a valid connection handle (`LDAP` structure) that you have created by calling `ldap_init()`. The function returns 0 if successful or 1 if an error occurred.

Code Example 11-1 Getting the root DSE attribute values

```
int printdse( LDAP *ld )
{
  int rc, i;
  char *matched_msg = NULL, *error_msg = NULL;
  LDAPMessage  *result, *e;
  BerElement  *ber;
  char    *a;
  char    **vals;
  char    *attrs[3];
```

Code Example 11-1 Getting the root DSE attribute values

```
/* Verify that the connection handle is valid. */
if ( ld == NULL ) {
  fprintf( stderr, "Invalid connection handle.\n" );
  return( 1 );
}
/* Set automatic referral processing off. */
if ( ldap_set_option( ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF ) != 0 ) {
  rc = ldap_get_lderrno( ld, NULL, NULL );
  fprintf( stderr, "ldap_set_option: %s\n", ldap_err2string( rc ) );
  return( 1 );
}
/* Search for the root DSE. */
attrs[0] = "supportedControl";
attrs[1] = "supportedExtension";
attrs[2] = NULL;
rc = ldap_search_ext_s( ld, "", LDAP_SCOPE_BASE, "(objectclass=*)", attrs,
  0, NULL, NULL, NULL, 0, &result );
/* Check the search results. */
switch( rc ) {
/* If successful, the root DSE was found. */
case LDAP_SUCCESS:
  break;
/* If the root DSE was not found, the server does not comply
   with the LDAPv3 protocol. */
case LDAP_PARTIAL_RESULTS:
case LDAP_NO_SUCH_OBJECT:
case LDAP_OPERATIONS_ERROR:
case LDAP_PROTOCOL_ERROR:
  printf( "LDAP server returned result code %d (%s).\n"
    "This server does not support the LDAPv3 protocol.\n",
    rc, ldap_err2string( rc ) );
  return( 1 );
/* If any other value is returned, an error must have occurred. */
default:
  fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
  return( 1 );
}
/* Since only one entry should have matched, get that entry. */
e = ldap_first_entry( ld, result );
if ( e == NULL ) {
  fprintf( stderr, "ldap_search_ext_s: Unable to get root DSE.\n");
  ldap_memfree( result );
  return( 1 );
}

/* Iterate through each attribute in the entry. */
for ( a = ldap_first_attribute( ld, e, &ber );
  a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {

  /* Print each value of the attribute. */
  if ((vals = ldap_get_values( ld, e, a)) != NULL ) {
    for ( i = 0; vals[i] != NULL; i++ ) {
      printf( "%s: %s\n", a, vals[i] );
    }
```

Code Example 11-1 Getting the root DSE attribute values

```
      /* Free memory allocated by ldap_get_values(). */
      ldap_value_free( vals );
    }

    /* Free memory allocated by ldap_first_attribute(). */
    ldap_memfree( a );
  }

  /* Free memory allocated by ldap_first_attribute(). */
  if ( ber != NULL ) {
    ber_free( ber, 0 );
  }

  printf( "\n" );
  /* Free memory allocated by ldap_search_ext_s(). */
  ldap_msgfree( result );
  ldap_unbind( ld );
  return( 0 );
}
```

Determining If the Server Supports LDAPv3

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,” on page 54 for details) before searching the directory. Unlike the LDAPv2protocol, the LDAPv3 protocol states that clients do not need to bind to the server before performing LDAP operations.

The following function connects to an LDAP server and determines whether or not that server supports the LDAPv3 protocol.

Code Example 11-2 Determining the supported LDAP version

```
/* Function for determining if the LDAP server supports LDAPv3.
  This function returns 1 if the server supports LDAPv3 or
  0 if the server does not support LDAPv3.
 */
int
check_version( char *hostname, int portnum )
{
```

Code Example 11-2 Determining the supported LDAP version

```
LDAP    *ld;
int    i, rc, v3supported = 0;
LDAPMessage  *result, *e;
BerElement  *ber;
LDAPControl  **serverctrls = NULL, **clntctrls = NULL;
char    *a, *dn;
char    **vals;
char    *attrs[2];
char    *filter = "(objectClass=*)";
/* Check arguments */
if ( !hostname || !hostname[0] || !portnum ) {
  printf( "Error: hostname or port number not specified\n" );
  return( -1 );
}
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( hostname, portnum )) == NULL ) {
  perror( "ldap_init" );
  return( -1 );
}
/* Set automatic referral processing off. */
if (ldap_set_option(ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF) != LDAP_SUCCESS) {
  ldap_perror( ld, "ldap_set_option" );
  return( -1 );
}
/* Search for the root DSE and get the supportedLDAPVersion attribute. */
attrs[0] = "supportedLDAPVersion";
attrs[1] = NULL;
rc = ldap_search_ext_s( ld, "", LDAP_SCOPE_BASE, filter, attrs, 0,
  serverctrls, clntctrls, NULL, 0, &result );
/* Check the search results. */
switch( rc ) {
/* If successful, the root DSE was found. */
case LDAP_SUCCESS:
  break;
/* If the root DSE was not found, the server does not comply
   with the LDAPv3 protocol. */
case LDAP_PARTIAL_RESULTS:
case LDAP_NO_SUCH_OBJECT:
case LDAP_OPERATIONS_ERROR:
case LDAP_PROTOCOL_ERROR:
  ldap_perror( ld, "ldap_search_ext_s" );
  return( 0 );
  break;
/* If an different result code is returned, an error may have
   occurred (for example, the server may be down. */
default:
  ldap_perror( ld, "ldap_search_ext_s" );
  return( -1 );
  break;
}
/* Get the values of the supportedLDAPVersion attribute in the entry. */
if (( e = ldap_first_entry( ld, result )) != NULL  &&
    ( a = ldap_first_attribute( ld, e, &ber )) != NULL  &&
    (vals = ldap_get_values( ld, e, a)) != NULL ) {
```

Code Example 11-2 Determining the supported LDAP version

```
    for ( i = 0; vals[i] != NULL; i++ ) {
      if ( !strcmp( "3", vals[i] ) ) {
        v3supported = 1;
        break;
      }
    }
    /* Free any memory allocated. */
    ldap_value_free( vals );
    ldap_memfree( a );
    if ( ber != NULL ) {
      ber_free( ber, 0 );
    }
  }
  /* Free memory allocated by ldap_search_ext_s(). */
  ldap_msgfree( result );
  /* Free the ld structure. */
  ldap_unbind_s( ld );
  /* Return a value indicating whether or not LDAPv3 is supported. */
  return( v3supported );
}
...
```

Getting Schema Information

In the LDAPv3 protocol, 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,” on page 100 for details.)

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 (this specifies the attributes that can be used with this extensible matching rule). Each value of this attribute is a matching rule use description.

For the format of the attribute values, reference *LDAPv3: Attrubute Syntax Definitions*, RFC 2252. For more information, see “Where to Find Additional Information” on page 21.

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 LDAP SDK for C
- Prerequisites for Connecting Over SSL
- Enabling Your Client to Connect Over SSL
- Installing Your Own SSL I/O Functions
- Using Certificate-Based Client Authentication

How SSL Works with the LDAP SDK for C

The Netscape LDAP SDK for C includes functions to enable your application to connect to an LDAP server over a Secure Sockets Layer (SSL). The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications.

The Netscape LDAP SDK for C only supports SSL 3.0 and does not support the Start Transport Layer Security (TLS) Operation. SSL communication must take place on a separate TCP port. Refer to “Where to Find Additional Information” on page 21 for information on SSL. Note that SSL is not supported by all LDAP servers.

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.

The Netscape LDAP SDK for C includes API functions that allow you to connect over SSL to an LDAP server. The API functions allow you to do the following:

- Set the session option for communicating with the server over SSL. For more information, see "Enabling Your Client to Connect Over SSL."
- Replace the default I/O functions with your own I/O functions for communicating over SSL. For more information, see "Installing Your Own SSL I/O Functions."
- Enable your client to send certificates to authenticate itself. For more information, see "Using Certificate-Based Client Authentication."

The API functions require 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".

Prerequisites for Connecting Over SSL

The API functions in the Netscape LDAP SDK for C that enable you to connect over SSL rely assume the following:

- Your client has access to a Netscape or iPlanet certificate database.

  This must be the `cert7.db` database file used by Netscape Communicator 4.*x* and above.

  Note that previous and later versions of these applications use different file formats for the certificate database. Attempting to use a different version of the certificate database will result in database errors.

  The LDAP API function call uses this certificate database to determine if it can 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 must be the `key3.db` file used by Netscape Communicator 4.*x* and greater).

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.

Enabling Your Client to Connect Over SSL

To connect to an LDAP server using SSL, you need to:

1. Initialize your client by calling one of the following functions:
   - Call `ldapssl_client_init()` if you do not plan to use certificate-based client authentication.
   - Call `ldapssl_clientauth_init()` if you plan to use certificate-based client authentication.

- Call `ldapssl_clientauth_init()` if you plan to use certificate-based client authentication and need to specify either the name and path of the security module database or the method of verifying the server's certificate.

2. Initialize an LDAP session with the secure server by calling the `ldapssl_init()` function.

Note that you need to initialize your client before initializing the LDAP session. The process of initializing the client opens the certificate database.

The following example initializes a client to connect to a secure LDAP server over SSL.

Code Example 12-1 Initializing a client SSL connection

```
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert7.db", NULL ) < 0) {
  printf( "Failed to initialize SSL client...\n" );
  return( 1 );
}
/* get a handle to an LDAP connection */
if ( (ld = ldapssl_init( "cert.netscape.com", LDAPS_PORT, 1 )) == NULL {
  perror( "ldapssl_init" );
  return( 1 );
}
...
/* Client can now perform LDAP operations on the secure LDAP server */
...
```

As an alternative to calling the `ldapssl_init()` function, you can do the following:

1. Initialize an LDAP session with the server by calling the standard initialization function `ldapssl_init()`.
2. Install the standard SSL I/O functions by calling `ldapssl_install_routines()`.
3. Set the SSL option in the LDAP struct by calling `ldap_set_option()`.

The effect of calling these three functions is the same as calling the `ldapssl_init()` function.

Note that you need to initialize your client before initializing the LDAP session. The process of initializing the client opens the certificate database.

The following example prepares a client to connect to a secure LDAP server over SSL.

Code Example 12-2 An alternate client SSL initialization

```
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert7.db", NULL ) < 0) {
  printf( "Failed to initialize SSL client...\n" );
  return( 1 );
}
/* Initialize LDAP session */
if ( (ld = ldap_init( MY_HOST, LDAPS_PORT )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Load SSL routines */
if ( ldapssl_install_routines( ld ) != 0 ) {
  ldap_perror( ld, "ldapssl_install_routines" );
  return( 1 );
}
/* Set up option in LDAP struct for using SSL */
if ( ldap_set_option( ld, LDAP_OPT_SSL, LDAP_OPT_ON ) != 0 ) {
  ldap_perror( ld, "ldap_set_option" );
  return( 1 );
}
/* Client can now perform LDAP operations on the secure LDAP server */
...
```

Handling Errors

The function `ldapssl_err2string()` provides support for special SSL error messages that are not handled by the normal error conversion routine `ldap_err2string()`. After calling any of the SSL initialization functions, you may convert SSL-specific errors codes to text strings by calling `ldapssl_err2string()`.

Installing Your Own SSL I/O Functions

The `ldapssl_init()` and `ldapssl_install_routines()` functions both set up the LDAP session to use the standard SSL I/O functions provided with the Netscape LDAP SDK for C.

If you want to use your own SSL I/O functions, you can use the `ldap_io_fns` structure. If you plan to specify your own SSL I/O functions, follow these steps:

1. Create an `ldap_io_fns` structure, and set the fields to point to your I/O functions.
2. Call `ldap_set_option()` to point to that structure.

For example:

Code Example 12-3 Setting custom SSL I/O functions

```
if (ldap_set_option( ld, LDAP_OPT_IO_FN_PTRS, &my_io_struct) != 0 ) {
  ldap_perror( ld, "ldap_set_option" );
  return( 1 );
}
```

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.

To configure your client to use certificates for authentication, follow these steps:

1. Initialize your client by calling one of the following functions:
   - `ldapssl_clientauth_init()`
   - `ldapssl_advclientauth_init()`

     Use this function if you need to specify the name and path of a security module database or if you need to specify the method used to verify the server's certificate.

   You should call one of these functions instead of the `ldapssl_client_init()` function to initialize your client.

   Note that you can also use these functions to initialize your client even if you do not plan to use certificate-based client authentication. These functions are equivalent to `ldapssl_client_init()`.
2. Initialize an LDAP session with the secure server by calling `ldapssl_init()`.
3. Enable your client to authenticate with the secure server by calling `ldapssl_enable_clientauth()`.

4. (Optional) Perform a SASL bind operation using the mechanism "EXTERNAL". This indicates to the directory server that certificates should be used to authenticate clients.

   In all releases of the iPlanet Directory Server and in Netscape Directory Server 4.*x* and later, if you perform a SASL bind operation and the server cannot find the corresponding directory entry for a client certificate, the server returns an `LDAP_INVALID_CREDENTIALS` result code with the error message "`client certificate mapping failed`."

Chapter 13

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
- Determining the SASL Mechanisms Supported
- Authenticating Using a SASL Mechanism

Understanding SASL

Simple Authentication and Security Layer (SASL) is described in RFC 2222, which you can find at this location:

`http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2222.txt`

SASL provides the means to use mechanisms other than simple authentication and SSL to authenticate to the Netscape LDAP SDK for C.

The ability to authenticate to an LDAP server using a SASL mechanism is a feature new to LDAPv3 (LDAPv2 servers do not support this method of authentication).

Determining the SASL Mechanisms Supported

To determine the SASL mechanisms supported by an LDAPv3 server, get the root DSE of the server, and check the `supportedSASLMechanisms` attribute. The values of this attribute are the names of the SASL mechanisms supported by the server.

If the root DSE does not have a `supportedSASLMechanisms` attribute, the server does not support any SASL mechanisms.

For information on getting the root DSE, see “Getting the Root DSE,” on page 208.

Authenticating Using a SASL Mechanism

To authenticate to the server using a SASL mechanism, call one of the following functions:

- The synchronous `ldap_sasl_bind_s()` function (see “Performing a Synchronous SASL Bind Operation,” on page 224).
- The asynchronous `ldap_sasl_bind()` function (see “Performing a Synchronous SASL Bind Operation”).

For more information about the difference between synchronous and asynchronous functions, see “Calling Synchronous and Asynchronous Functions,” on page 72.

If you call the asynchronous function `ldap_sasl_bind()`, you need to call the `ldap_result()` and `ldap_parse_sasl_bind_result()` functions to get the result of the SASL bind operation.

Authentication using a SASL mechanism may take one or more round trips between your LDAP client and the server. (The server may send a number of "challenges" to the client.) You may need to call `ldap_sasl_bind_s()` several times (or `ldap_sasl_bind()`, `ldap_result()`, and `ldap_parse_sasl_bind_result()` several times) in order to respond to each server challenge.

Before calling the function to perform a SASL bind operation, make sure to specify that your client is LDAPv3 compliant. If you do not, an `LDAP_NOT_SUPPORTED` result code is returned. For details, see “Specifying the LDAP Version of Your Client,” on page 52.

Performing a Synchronous SASL Bind Operation

If you want to wait for the results of the SASL bind operation to complete before continuing, call the synchronous `ldap_sasl_bind_s()` function. This function sends a SASL bind request to the server and blocks until the server sends the results of the operation back to your client.

The `ldap_sasl_bind_s()` function returns one of the following values:

- `LDAP_SUCCESS` if your client has successfully authenticated.
- `LDAP_SASL_BIND_IN_PROGRESS` if the server sends a challenge to your client. If you receive this result code, check the `servercredp` argument for the `berval` structure containing the server's challenge. Call the `ldap_sasl_bind_s()` function again to send a response to that challenge.
- An LDAP error code, if a problem occurred or if authentication failed.

See the documentation on the `ldap_sasl_bind_s()` function for a list of the possible result codes.

Performing an Asynchronous SASL Bind Operation

If you want to perform other work (in parallel) while waiting the SASL bind operation to complete, do the following:

1. To send an LDAP SASL bind request, call the asynchronous `ldap_sasl_bind()` function.

   This function returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred while sending the request). The function also sets the `msgidp` argument to point to a message ID identifying the SASL bind operation.

2. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID.

   The `ldap_result()` function uses the message ID to determine if the server sent a SASL bind response. The function passes back the response in an `LDAPMessage` structure.

3. Call the `ldap_parse_sasl_bind_result()` function to parse the `LDAPMessage` structure and retrieve information from the server's response.

   If the server sent a challenge to your client, the challenge is specified in the `berval` structure passed back as the `servercredp` argument.

4. Call the `ldap_get_lderrno()` function to get the LDAP result code for the operation. The function can return one of the following values:

   - `LDAP_SUCCESS` if your client successfully authenticated to the server.
   - `LDAP_SASL_BIND_IN_PROGRESS` if the server sent a challenge to your client.

- An LDAP error code, if a problem occurred or if authentication failed.

  See the documentation on the return values of the `ldap_sasl_bind_s()` function for a list of result codes that the server can return for this operation.

  If the server returned an `LDAP_SASL_BIND_IN_PROGRESS` result code, check the `servercredp` argument for the `berval` structure containing the server's challenge.

5. If the result code is `LDAP_SASL_BIND_IN_PROGRESS` and the server passed back another challenge, determine the response to that challenge and call the `ldap_sasl_bind()` function again to send that response to the server.

   You can call e `ldap_result()` and `ldap_parse_sasl_bind_result()` again to get the next challenge sent from the server, if the result is `LDAP_SASL_BIND_IN_PROGRESS` again.

The following example is an LDAP client that authenticates using the SASL mechanism named `babsmechanism`.

Code Example 13-1 Authenticating over SASL

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "ldap.h"

int
main( int argc, char **argv )
{
    LDAP        *ld;
    LDAPMod     mod0;
    LDAPMod     mod1;
    LDAPMod     *mods[ 3 ];
    char        *vals0[ 2 ];
    char        *vals1[ 2 ];
    time_t      now;
    char        buf[ 128 ];
    struct berval   cred;
    struct berval   *servcred;
    int         version;
    /* get a handle to an LDAP connection */
    if ( (ld = ldap_init( "localhost", 389 )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
    }
    /* Set the LDAP protocol version supported by the client
       to 3. (By default, this is set to 2. SASL authentication
       is part of version 3 of the LDAP protocol.) */
    version = LDAP_VERSION3;
    ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
```

Code Example 13-1 Authenticating over SASL

```
    /* authenticate */
    cred.bv_val = "magic";
    cred.bv_len = sizeof( "magic" ) - 1;
    if ( ldap_sasl_bind_s( ld, "uid=bjensen,ou=people,o=airius.com", \
      "babsmechanism", &cred, NULL, NULL, &servcred ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_sasl_bind_s" );
  return( 1 );
    }
    /* get and print the credentials returned by the server */
    printf( "Server credentials: %s\n", servcred->bv_val );
    /* construct the list of modifications to make */
    mod0.mod_op = LDAP_MOD_REPLACE;
    mod0.mod_type = "mail";
    vals0[0] = "babs@airius.com";
    vals0[1] = NULL;
    mod0.mod_values = vals0;
    mod1.mod_op = LDAP_MOD_ADD;
    mod1.mod_type = "description";
    time( &now );
    sprintf( buf, "This entry was modified with the modattrs program on %s",
      ctime( &now ));
    /* Get rid of \n which ctime put on the end of the time string */
    if ( buf[ strlen( buf ) - 1 ] == '\n' ) {
  buf[ strlen( buf ) - 1 ] = '\0';
    }
    vals1[ 0 ] = buf;
    vals1[ 1 ] = NULL;
    mod1.mod_values = vals1;
    mods[ 0 ] = &mod0;
    mods[ 1 ] = &mod1;
    mods[ 2 ] = NULL;
    /* make the change */
    if ( ldap_modify_s( ld, "uid=bjensen,ou=people,o=airius.com", mods )
      != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_modify_s" );
  return( 1 );
    }
    ldap_unbind( ld );
    printf( "modification was successful\n" );
    return( 0 );
}
```

Chapter 14

Working with LDAP Controls

This chapter explains how LDAP controls work and how to use the LDAP controls that are supported by the Netscape LDAP SDK for C, v4.1.

The chapter includes the following sections:

- How LDAP Controls Work
- Using Controls in the LDAP API
- Determining the Controls Supported By the Server
- Using the Server-Side Sorting Control
- Using the Persistent Search Control
- Using the Entry Change Notification Control
- Using the Virtual List View Control
- Using the Manage DSA IT Control
- Using Password Policy Controls
- Using the Proxied Authorization Control

How LDAP Controls Work

The LDAPv3 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 iPlanet Directory Server 5.0 and later and Netscape Directory Server 4.*x* send 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), as defined by the creator of this control.
- 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 instead return the result code `LDAP_UNAVAILABLE_CRITICAL_EXTENSION`.
    - If the control is marked as not critical to the operation, the server should ignore the control and should perform the operation.

Note that servers can also send controls back to clients.

The LDAP API supports 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 API only and are never sent to the server.

The next section describes how controls are implemented in the LDAP API and which functions you can call to create, send, and parse data from LDAP controls.

Using Controls in the LDAP API

In the LDAP API, a control is represented by an `LDAPControl` structure:

```
typedef struct ldapcontrol {
  char                *ldctl_oid;
  struct berval       ldctl_value;
  char                ldctl_iscritical;
} LDAPControl;
```

The fields in this structure represent the data in a control:

- `ldctl_oid` specifies the OID of the control.
- `ldctl_value` contains a `berval` structure containing data associated with the control.
- `ldctl_iscritical` specifies whether or not the control is critical to the operation (`LDAP_OPT_ON` indicates that the control is critical, and `LDAP_OPT_OFF` indicates that the control is not critical).

For more information on the `LDAPControl` structure, see Chapter 17, “Data Types and Structures.”

You can either allocate and create the control yourself, or you can call an LDAP API function to create the control. For example, you can call the `ldap_create_sort_control()` function to create a server-sorting control.

To include a control in a request, call one of the LDAPv3 API functions (functions with names ending with `_ext` and `_ext_s`). These functions allow you to pass in an array of server controls and an array of client controls.

(You can also include controls in a request by specifying the array of controls in the `LDAP_OPT_SERVER_CONTROLS` option. Note, however, that these controls will be sent to the server with every request. If the control is specific to a certain type of operation, you should use the `_ext` and `_ext_s` functions instead.)

To retrieve any controls included in a server’s response, call the `ldap_parse_result()` function. You can then retrieve data from the returned controls yourself (by checking the fields of the `LDAPControl` structure) or by calling additional API functions (such as the `ldap_parse_sort_control()` function).

When you are done working with a control or with an array of controls, you should free them from memory by calling the `ldap_control_free()` function or the `ldap_controls_free()` function.

The rest of this chapter explains how to determine which controls are supported by an LDAPv3 server and how to use LDAP API functions to send and retrieve specific types of controls.

Determining the Controls Supported By the Server

According to the LDAPv3 protocol, servers should list any controls that they support in the `supportedControl` attribute in the root DSE. (See “Understanding DSEs,” on page 207 and “Getting the Root DSE,” on page 208 for more information.)

The following table lists some of the OIDs for server controls.

Table 14-1 LDAPv3 Server Controls

OID of Control	Defined Name (in ldap.h)	Description of Control
2.16.840.1.113730.3.4.2	`LDAP_CONTROL_MANAGEDSAIT`	"Manage DSA IT" control (see “Using the Manage DSA IT Control” for details).
2.16.840.1.113730.3.4.3	`LDAP_CONTROL_PERSISTENTSEARCH`	"Persistent search" control (see “Using the Persistent Search Control” for details).
2.16.840.1.113730.3.4.4	`LDAP_CONTROL_PWEXPIRED`	"Password expired" control (see “Using Password Policy Controls” for details).
2.16.840.1.113730.3.4.5	`LDAP_CONTROL_PWEXPIRING`	"Password expiration warning" control (see “Using Password Policy Controls” for details).
2.16.840.1.113730.3.4.9	`LDAP_CONTROL_VLVREQUEST`	"Virtual list view" control (see “Using the Virtual List View Control” for details).
1.2.840.113556.1.4.473	`LDAP_CONTROL_SORTREQUEST`	"Server-side sorting" control (see “Using the Server-Side Sorting Control” for details).

Table 14-1 LDAPv3 Server Controls

OID of Control	Defined Name (in ldap.h)	Description of Control
2.16.840.1.113730.3.4.12	LDAP_CONTROL_PROXYAUTH	"Proxy authorization" control (see “Using the Proxied Authorization Control” for details).

The following example is a simple command-line program that searches for the root DSE and prints the values of the supported Control attribute.

Code Example 14-1 Searching the root DSE and outputting the Control attribute

```
#include "ldap.h"
static char *usage = "Usage: listctrl -h <hostname> -p <portnumber>\n";

/* Associate OIDs of known controls with descriptions. */
struct oid2desc {
  char  *oid;
  char  *desc;
};
static struct oid2desc oidmap[] = {
  {LDAP_CONTROL_MANAGEDSAIT,    "Manage DSA IT control"},
  {LDAP_CONTROL_SORTREQUEST,    "Server-side sorting control"},
  {LDAP_CONTROL_PERSISTENTSEARCH,   "Persistent search control"},
  {LDAP_CONTROL_VLVREQUEST,    "Virtual list view control"},
  {LDAP_CONTROL_PWEXPIRED,    "Password expired control"},
  {LDAP_CONTROL_PWEXPIRING,    "Password expiration warning control"},
  { NULL, NULL }
};
int
main( int argc, char **argv )
{
  LDAP          *ld;
  LDAPMessage   *result, *e;
  char          *hostname = NULL;
  char          **vals;
  char          *attrs[2];
  int           i, j, c, portnumber = LDAP_PORT, rc;
  LDAPControl   **serverctrls = NULL, **clntctrls = NULL;
  /* Parse the command line arguments. */
  while ( ( c = getopt( argc, argv, "h:p:" ) ) != -1 ) {
    switch ( c ) {
    case 'h':
      hostname = strdup( optarg );
      break;
    case 'p':
      portnumber = atoi( optarg );
      break;
    default:
      printf( "Unsupported option: %c\n", c );
      printf( usage );
```

Code Example 14-1 Searching the root DSE and outputting the Control attribute

```
      exit( 1 );
    }
  }
  /* By default, connect to localhost at port 389. */
  if ( hostname == NULL || hostname[0] == NULL ) {
      hostname = "localhost";
  }
  /* Initialize the connection. */
  if ( (ld = ldap_init( hostname, portnumber )) == NULL ) {
      perror( "ldap_init" );
      return( 1 );
  }
  /* Set automatic referral processing off. */
  if ( ldap_set_option( ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF )
    != LDAP_SUCCESS ) {
      ldap_perror( ld, "ldap_set_option" );
      return( 1 );
  }
  /* Search for the root DSE and retrieve only the
    supportedControl attribute. */
  attrs[ 0 ] = "supportedControl";
  attrs[ 1 ] = NULL;
  rc = ldap_search_ext_s( ld, "", LDAP_SCOPE_BASE, "(objectclass=*)",
    attrs, 0, serverctrls, clntctrls, NULL, NULL, &result );
  /* Check the search results. */
  switch( rc ) {
  /* If successful, the root DSE was found. */
  case LDAP_SUCCESS:
    break;
  /* If the root DSE was not found, the server does not comply
    with the LDAPv3 protocol. */
  case LDAP_PARTIAL_RESULTS:
  case LDAP_NO_SUCH_OBJECT:
  case LDAP_OPERATIONS_ERROR:
  case LDAP_PROTOCOL_ERROR:
    printf( "LDAP server %s:%d returned result code %d (%s).\n"
    "This server does not support the LDAPv3 protocol.\n",
    hostname, portnumber, rc, ldap_err2string( rc ) );
    return( 1 );
    break;
  /* If any other value is returned, an error must have occurred. */
  default:
    ldap_perror( ld, "ldap_search_ext_s" );
    return( 1 );
    break;
  }
  /* Get the root DSE from the results.
    Since there is only one root DSE, there
    should be only one entry in the results. */
  e = ldap_first_entry( ld, result );
  /* Get and print the values of the supportedControl attribute. */
  if (e != NULL &&
    (vals = ldap_get_values(ld, e, "supportedControl")) != NULL ) {
    printf( "\nControls Supported by %s:%d\n", hostname, portnumber );
    printf( "==================================================\n" );
```

Code Example 14-1 Searching the root DSE and outputting the Control attribute

```
    for ( i = 0; vals[i] != NULL; i++ ) {
      printf( "%s\n", vals[i] );
      for ( j = 0; oidmap[j].oid != NULL; j++ ) {
        if ( !strcmp( vals[i], oidmap[j].oid )) {
          printf( "\t%s\n", oidmap[j].desc );
        }
      }
    }
    /* Free the values allocated by ldap_get_values(). */
    ldap_value_free( vals );
    printf( "\n" );
  }
  /* Free memory allocated by ldap_search_ext_s(). */
  ldap_msgfree( result );
  ldap_unbind( ld );
  return( 0 );
}
```

Using the Server-Side Sorting Control

The control with the OID 1.2.840.113556.1.4.473 (or `LDAP_CONTROL_SORTREQUEST`, as defined in the `ldap.h` header file) 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 RFC 2891: `ftp://ftp.isi.edu/in-notes/rfc2891.txt`

The following sections explain how to use the server-side sorting control:

- Specifying the Sort Order
- Creating the Control
- Performing the Search
- Interpreting the Results
- Example of Using the Server-Sorting Control

Specifying the Sort Order

To specify the sort order of the results, you can call the `ldap_create_sort_keylist()` function. This function creates a sort key list from a string in the following format:

`[-]<`*attrname*`>[:<`*matching_rule_oid*`>] ...`

`attrname` is the name of the attribute that you want to sort by. You can specify a space-delimited list of attribute names. `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 last name ("`sn`") first in ascending order. If multiple entries have the same last name, these entries are sorted by first name ("`givenname`") in descending order:

```
"sn -givenname"
```

Passing this string to the `ldap_create_sort_keylist()` function creates a sort key list, which is an array of `LDAPsortkey` structures. You can use this to create the server-side sorting control.

Creating the Control

Next, to create the server-side sorting control, you pass the sort key list (the array of `LDAPsortkey` structures) to the `ldap_create_sort_control()` function.

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 ramifications of marking the control as critical.

The function passes back a newly created sort control, an `LDAPControl` structure, which you can include in a search request.

After you call the `ldap_create_sort_control()` function and create the control, you should free the array of `LDAPsortkey` structures by calling `ldap_free_sort_keylist()`.

When you are done receiving sorted results from the server, you should free the `LDAPControl` structure by calling `ldap_control_free()`.

Performing the Search

To specify that you want the server to sort the results, add the newly created server-sorting control to a `NULL`-terminated array of `LDAPControl` structures and pass this array to the `ldap_search_ext()` function or the `ldap_search_ext_s()` function.

The server returns a result for the search operation and a response control. The response control indicates the success or failure of the sorting. To determine if sorting was successful, do the following:

1. Call `ldap_parse_result()` to parse the result of the search operation and retrieve any response controls sent back from the server.

   Response controls are passed back in a `NULL`-terminated array of `LDAPControl` structures.

2. Pass this array of structures as an argument to `ldap_parse_sort_control()` to retrieve the LDAP result code for the sorting operation.

   If the sorting operation fails, the server may also return the name of the attribute that caused the failure. The `ldap_parse_sort_control()` function also retrieves this name, if available.

When you are done parsing the array of response controls, you should free the array by calling the `ldap_controls_free()` function.

The server can return the following result codes that apply to the sorting operation.

Table 14-2 LDAP result codes for sorting search results

Result Code	Description
`LDAP_SUCCESS`	The results were sorted successfully.
`LDAP_OPERATIONS_ERROR`	An internal server error occurred.
`LDAP_TIMELIMIT_EXCEEDED`	The maximum time allowed for a search was exceeded before the server finished sorting the results.
`LDAP_STRONG_AUTH_REQUIRED`	The server refused to send back the sorted search results because it requires you to use a stronger authentication method.
`LDAP_ADMINLIMIT_EXCEEDED`	There are too many entries for the server to sort.
`LDAP_NO_SUCH_ATTRIBUTE`	The sort key list specifies an attribute that does not exist.
`LDAP_INAPPROPRIATE_MATCHING`	The sort key list specifies a matching rule that is not recognized or appropriate.
`LDAP_INSUFFICIENT_ACCESS`	The server did not send the sorted results because the client has insufficient access rights.
`LDAP_BUSY`	The server is too busy to sort the results.
`LDAP_UNWILLING_TO_PERFORM`	The server is unable to sort the results.

Table 14-2 LDAP result codes for sorting search results

Result Code	Description
LDAP_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.

Table 14-3 Server responses to sorting controls under different circumstances

Does the Server Support the Sort Control?	Is the Sort Control Marked As Critical?	Other Conditions	Results from LDAP Server
No	Yes	N / A	The server does not send back any entries.
	No		The server ignores the sorting control and returns the entries unsorted.
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.
	No		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.
	N / A (may or may not be marked as critical)	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 (LDAP_SUCCESS).
		The search itself failed (for any reason).	The server sends back a result code for the search operation.
			The server does not send back the sorting response control.

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.

Code Example 14-2 Applying the server-sorting control

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER 3890
int
main( int argc, char **argv )
{
  LDAP         *ld;
  LDAPMessage  *result, *e;
  char         *attrfail, *matched = NULL, *errmsg = NULL;
  char         **vals, **referrals;
  int          rc, parse_rc, version;
  unsigned long     rcode;
  LDAPControl       *sortctrl = NULL;
  LDAPControl       *requestctrls[ 2 ];
  LDAPControl       **resultctrls = NULL;
  LDAPsortkey       **sortkeylist;
  /* Get a handle to an LDAP connection */
  if ( (ld = ldap_init( HOSTNAME, PORTNUMBER ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
  version = LDAP_VERSION3;
  ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
  /* Create a sort key list that specifies the sort order of the results.
     Sort the results by last name first, then by first name. */
  ldap_create_sort_keylist( &sortkeylist, "sn -givenname" );
  /* Create the sort control. */
  rc = ldap_create_sort_control( ld, sortkeylist, 1, &sortctrl );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_create_sort_control: %s\n", ldap_err2string( rc )
);
    ldap_unbind( ld );
    return( 1 );
  }
  requestctrls[ 0 ] = sortctrl;
  requestctrls[ 1 ] = NULL;
  /* Search for all entries in Sunnyvale */
  rc = ldap_search_ext_s( ld, "o=airius.com", LDAP_SCOPE_SUBTREE,
    "(mail=*airius.com*)", NULL, 0, requestctrls, NULL, NULL, 0, &result );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
```

Code Example 14-2 Applying the server-sorting control

```
  parse_rc = ldap_parse_result( ld, result, &rc, &matched, &errmsg,
&referrals, &resultctrls, 0 );
  if ( parse_rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc ) );
    ldap_unbind( ld );
    return( 1 );
  }
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    if ( errmsg != NULL && *errmsg != '\0' ) {
      fprintf( stderr, "%s\n", errmsg );
    }
    ldap_unbind( ld );
    return( 1 );
  }
  parse_rc = ldap_parse_sort_control( ld, resultctrls, &rcode, &attrfail );
  if ( parse_rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_parse_sort_control: %s\n", ldap_err2string(
parse_rc ) );
    ldap_unbind( ld );
    return( 1 );
  }

  if ( rcode != LDAP_SUCCESS ) {
    fprintf( stderr, "Sort error: %s\n", ldap_err2string( rcode ) );
    if ( attrfail != NULL && *attrfail != '\0' ) {
      fprintf( stderr, "Bad attribute: %s\n", attrfail );
    }
    ldap_unbind( ld );
    return( 1 );
  }
  /* for each entry print out name + all attrs and values */
  for ( e = ldap_first_entry( ld, result ); e != NULL;
      e = ldap_next_entry( ld, e ) ) {
    if ((vals = ldap_get_values( ld, e, "sn")) != NULL ) {
      if ( vals[0] != NULL ) {
        printf( "%s", vals[0] );
      }
      ldap_value_free( vals );
    }
    if ((vals = ldap_get_values( ld, e, "givenname")) != NULL ) {
      if ( vals[0] != NULL ) {
        printf( "\t%s", vals[0] );
      }
      ldap_value_free( vals );
    }
    printf( "\n" );
  }
  ldap_msgfree( result );
  ldap_free_sort_keylist( sortkeylist );
  ldap_control_free( sortctrl );
  ldap_controls_free( resultctrls );
  ldap_unbind( ld );
```

Code Example 14-2 Applying the server-sorting control

```
  return( 0 );
}
```

Using the Persistent Search Control

The control with the OID 2.16.840.1.113730.3.4.3 (or `LDAP_CONTROL_PERSISTENTSEARCH`, as defined in the `ldap.h` header file) 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-Drafts *Persistent Search: A Simple LDAP Change Notification Mechanism* and *LDAP C API Extensions for Persistent Search*. For details, see “Where to Find Additional Information” on page 21.

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.”

To create a persistent search control, call `ldap_create_persistentsearch_control()`.

```
int ldap_create_persistentsearch_control( LDAP *ld,
  int changetypes, int changesonly, int return_echg_ctls,
  char ctl_iscritical, LDAPControl **ctrlp );
```

You can specify the following information:

- `changetypes` specifies 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):
  - `LDAP_CHANGETYPE_ADD` indicates that you want to track added entries.
  - `LDAP_CHANGETYPE_DELETE` indicates that you want to track deleted entries.
  - `LDAP_CHANGETYPE_MODIFY` indicates that you want to track modified entries.

- LDAP_CHANGETYPE_MODDN indicates that you want to track renamed entries.
- LDAP_CHANGETYPE_ANY indicates that you want to track all changes to entries.

- changesonly indicates whether or not you want the server to return all entries that initially matched the search criteria (0 to return all entries, or non-zero to return only the entries that change).
- return_echg_ctls indicates whether or not you want entry change notification controls included with every modified entry returned by the server (non-zero to return entry change notification controls).

The function passes back an LDAPControl structure representing the control in the ctrlp parameter. You can add the newly created control to a NULL-terminated array of LDAPControl structures and pass this array to the ldap_search_ext() function.

To end the persistent search, you can either call the ldap_abandon_ext() function to abandon the search operation, or you can call the ldap_unbind() function to disconnect from the server.

For an example showing how to perform a persistent search, refer to the example provided with the LDAP SDK for C, psearch.c.

Using the Entry Change Notification Control

The control with the OID 2.16.840.1.113730.3.4.7 (or LDAP_CONTROL_ENTRYCHANGE, as defined in the ldap.h header file) is the "entry change notification" control. This control contains additional information about the change made to the entry, including the type of change made, the change number (which corresponds to an item in the server's change log, if the server supports a change log), and, if the entry was renamed, the old DN of the entry.

The entry change notification control is described in the Internet-Drafts *Persistent Search: A Simple LDAP Change Notification Mechanism* and *LDAP C API Extensions for Persistent Search*. For more information on these documents, see “Where to Find Additional Information” on page 21.

You use this control in conjunction with a persistent search control. (See “Using the Persistent Search Control.”) If you have specified the preference for returning entry change notification controls, the server includes an entry change notification control with each entry found by the search.

To retrieve and parse an entry change notification control included with an entry, do the following:

1. Pass the `LDAPMessage` structure that represents an entry to the `ldap_get_entry_controls()` function.
2. Pass the entry change notification control to the `ldap_parse_entrychange_control()` function.

Using the Virtual List View Control

The control with the OID 2.16.840.1.113730.3.4.9 (or `LDAP_CONTROL_VLVREQUEST`, as defined in the `ldap.h` header file) 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. This version of the LDAP SDK for C supports this control.

The virtual list view control is described in the Internet-Draft *LDAP Extensions for Scrolling View Browsing of Search Results*. For details, see "Where to Find Additional Information" on page 21.

Using the Manage DSA IT Control

The control with the OID 2.16.840.1.113730.3.4.2 (or `LDAP_CONTROL_MANAGEDSAIT`, as defined in the `ldap.h` header file) 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*. For details, see "Where to Find Additional Information" on page 21.

To create this control, create an `LDAPControl` structure and set the `ldctl_oid` field to 2.16.840.1.113730.3.4.2.

When you add this control to the array of `LDAPControl` structures that you pass to a function (for example, `ldap_search_ext()` or `ldap_modify_ext()`, 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 `LDAP_CONTROL_PWEXPIRED`, as defined in the `ldap.h` header file) 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 `LDAP_CONTROL_PWEXPIRING`, as defined in the `ldap.h` header file) 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. The `ldctl_value` field of the `LDAPControl` structure specifies the number of seconds before the password will expire.

To get these server response controls when binding, you can do the following:

1. Call `ldap_simple_bind()` to send a request for an asynchronous bind operation.
2. Call `ldap_result()` to get the results of the operation.
3. Call the `ldap_parse_result()` function to parse the result and retrieve the server response controls from the result as an array of `LDAPControl` structures.

You can then check the `ldctl_oid` field to determine the OID of the control and the `ldctl_value` field for any data included in the control.

Using the Proxied Authorization Control

Proxied Authorization is an extension to the LDAPv3 protocol that allows a bound client to assume the identity of another directory entity without rebinding. This allows the client to perform operations as if it were bound as the proxied directory entity. All directory access (read, write, search, compare, delete, and add) is supported by proxied authorization.

This control is supported by Netscape Directory Server 4.1 or later and all versions of the iPlanet Directory Server. Proxied authorization support is based on the IETF Internet Draft. For more information, see “Where to Find Additional Information” on page 21.

As an example of proxied authorization, suppose a client is bound as `uid=bjensen, ou=Engineering, o=airius.com`. Further suppose the `bjensen` does not have the right to search the `ou=Marketing, o=airius.com` tree. However, `uid=lboyd, ou=Marketing, o=airius.com` does have rights to search the `Marketing` tree.

Further, `lboyd` grants proxy right to `bjensen`. In this case, `bjensen` may bind as herself, assume the identity of `lboyd`, and then search the `Marketing` tree.

This feature is intended as a performance and administrative benefit for certain types of directory usage. Specifically, applications that want to allow many clients to access or modify specific directory data without rebinding as another directory entity may want to use this feature.

Access Control: The Proxy Right

Proxied authorization adds an additional access right: proxy. If an entry grants the proxy right, then the entity to which that right is granted may assume the identity of the granting entity.

For example, if you wanted to allow `uid=bjensen` the right to proxy as `uid=lboyd`, add the following aci to your directory:

```
aci: (target = "ldap:///uid=lboyd, ou=Marketing,
  o=Airius.com")(targetattr=*)(version 3.0;
  aci "grant bjensen the right to proxy as lboyd";
  allow(proxy) userdn="ldap:///uid=bjensen,
  ou=Engineering, o=Airius.com";)
```

This aci allows `bjensen` to assume the identity of `lboyd` for all directory operations. Essentially, `bjensen` can do to the directory whatever `lboyd` has permission to do.

While the Netscape and iPlanet Directory Servers support this feature, be aware that other directory servers might not support this feature as described here. iPlanet Directory Server access control (including the proxy right) is fully described in the *iPlanet Directory Server Administrator's Guide.*

The Proxied Authorization Control

To support proxied authorization (an extension to the LDAPv3 protocol), the proxy authorization control has been added to the LDAP SDK for C in the form of the `ldap_create_proxyauth_control()` function. You use this function to create the control that allows a bound entity to assume the identity of another directory entry.

Proxy authorization is an optional LDAP server feature; it may not be supported on all LDAP servers. You should call the proxy authorization control function only when interacting with LDAP servers that support this LDAPv3 extension. You can check on the support of this control by looking at the rootDSE `supportedControl` attrubute. For example, the following command uses the ldapsearch utility to display the rootDSE:

`ldapsearch -v -h` *hostname* `-p 389 -b "" -s base ""`

In order for the control to work, the LDAP server that you are connecting to must support the server control for proxied authorization (OID 2.16.840.1.113730.3.4.12, or `LDAP_CONTROL_PROXYAUTH` as defined in the `ldap.h` header file).

Example

The following code fragment creates an LDAP connection, sets the proxied authorization control, binds to the directory, and then performs a search operation using the proxied authorization control. A more complete example is also available with the SDK example files.

Code Example 14-3 Proxied authorization control example

```
#include <ldap.h>

int             version;
LDAP            *ld;
LDAPControl     *requestctrls[ 2 ];
LDAPControl     *pactrl = NULL;

/* Customize the following host and bind information for your site. */
int         port=389;
char        *host="directory.airius.com";
char        *baseDN="o=airius.com";
```

Code Example 14-3 Proxied authorization control example

```
/* Proxied auth specific information.
   proxyDN is the entity that will be proxied.
   bindDN and bindpw is for the bind entity that will use the proxyDN. */
char            *proxyDN = "uid=lboyd, ou=marketing, o=airius.com";
char            *bindDN = "uid=bjensen, ou=engineering, o=airius.com";
char            *bindpw = "password";

/* Do general LDAP init stuff */
/* Get a handle to an LDAP connection */
if ( (ld = ldap_init( host, port ) ) == NULL ) {
        printf("ldap_init did not return a conn handle.\n");
        return;
        }
/* set version to ldap version 3 */
version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

/* authenticate to the directory */
if ( ldap_simple_bind_s( ld, bindDN, bindpw ) != LDAP_SUCCESS ) {
        printf("ldap_simple_bind_s failed");
        return (-1);
}

/* create the proxied authorization control */
if ( ldap_create_proxyauth_control( ld, proxyDN, 1, &pactrl ) ) {
   printf("ldap_create_proxyauth_control failed.\n");
   if ( ldap_unbind( ld ) != LDAP_SUCCESS ) {
        printf("ldap_unbind failed\n");
   }
   return(-1);
}

requestctrls[ 0 ] = pactrl;
requestctrls[ 1 ] = NULL;

/* Perform the search using the control */
printf("Searching for %s with the proxy auth control.\n", proxyDN);
if ( ldap_search_ext_s( ld, proxyDN, LDAP_SCOPE_SUBTREE, "(objectclass=*)",
        NULL, 0, requestctrls, NULL, NULL, LDAP_NO_LIMIT, &results ) !=
        LDAP_SUCCESS ) {
                printf("ldap_search_ext failed.\n");
                printf("Something is wrong with proxied auth.\n");
} else {
         print_search_results(ld, results);
}
```

Working with Extended Operations

This chapter explains how LDAPv3 extended operations work and how to use the extended operations that are supported by your LDAP server.

How Extended Operations Work

Extended operations are part of the LDAPv3 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.

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," on page 208.

Performing an Extended Operation

To perform an extended operation, call one of the following functions:

- The synchronous `ldap_extended_operation_s()` function (see "Performing a Synchronous Extended Operation").
- The asynchronous `ldap_extended_operation()` function (see "Performing an Asynchronous Extended Operation").

For more information about the difference between synchronous and asynchronous functions, see "Calling Synchronous and Asynchronous Functions," on page 72.

Both of these functions allow you to specify the OID of the extended operation and the data that you want applied to the operation.

Before calling the function to perform a LDAP extended operation, make sure to specify that your client is using version 3 of the LDAP protocol. If you do not, an `LDAP_NOT_SUPPORTED` result code is returned. For details, see "Specifying the LDAP Version of Your Client," on page 52.

Performing a Synchronous Extended Operation

If you want to wait for the results of an LDAP extended operation to complete before continuing, call the synchronous `ldap_extended_operation_s()` function. This function sends a SASL bind request to the server and blocks until the server sends the results of the operation back to your client.

The `ldap_extended_operation_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred. See the documentation on the `ldap_extended_operation_s()` function for a list of the possible result codes.

Performing an Asynchronous Extended Operation

If you want to perform other work (in parallel) while waiting an LDAP extended operation to complete, do the following:

1. To send an LDAP extended operation request, call the asynchronous `ldap_extended_operation()` function.

   This function returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred while sending the request). The function also sets the `msgidp` argument to point to a message ID identifying the extended operation.

2. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID.

   The `ldap_result()` function uses the message ID to determine if the server sent an extended operation response. The function passes back the response in an `LDAPMessage` structure.

3. Call the `ldap_parse_extended_result()` function to parse the `LDAPMessage` structure and retrieve information from the server's response.

   If the server sent an OID of an extended operation to your client, the OID is passed back as the `retoidp` argument.

   If the server sent a data to your client, the data is specified in the berval structure passed back as the `retdatap` argument.

4. Call the `ldap_get_lderrno()` function to get the LDAP result code for the operation. The function returns an `LDAP_SUCCESS` result code if the extended operation was performed successfully or an LDAP error code if a problem occurred.

   See the documentation on the return values of the function `ldap_extended_operation_s()` for a list of result codes that the server can return for this operation.

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.

Code Example 15-1 Requesting an extended operation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "ldap.h"

/* Name and port of the LDAP server you want to connect to. */
#define MY_HOST "localhost"
#define MY_PORT  389
/* DN of user (and password of user) who you want to authenticate as */
#define MGR_DN  "cn=Directory Manager"
#define MGR_PW  "23skidoo"
int
main( int argc, char **argv )
{
    /* OID of the extended operation that you are requesting */
    const char     *oidrequest = "1.2.3.4";
    char           *oidresult;
    struct berval  valrequest;
    struct berval  *valresult;
    LDAP           *ld;
    int rc, version;
    /* Set up the value that you want to pass to the server */
    printf( "Setting up value to pass to server...\n" );
    valrequest.bv_val = "My Value";
    valrequest.bv_len = strlen( "My Value" );
    /* Get a handle to an LDAP connection */
    printf( "Getting the handle to the LDAP connection...\n" );
    if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL ) {
  perror( "ldap_init" );
  ldap_unbind( ld );
  return( 1 );
    }
    /* Set the LDAP protocol version supported by the client
       to 3. (By default, this is set to 2. Extended operations
       are part of version 3 of the LDAP protocol.) */
    ldap_get_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
    printf( "Resetting version %d to 3.0...\n", version );
    version = LDAP_VERSION3;
    ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
    /* Authenticate to the directory as the Directory Manager */
    printf( "Binding to the directory...\n" );
    if ( ldap_simple_bind_s( ld, MGR_DN, MGR_PW ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_simple_bind_s" );
  ldap_unbind( ld );
  return( 1 );
    }
    /* Initiate the extended operation */
    printf( "Initiating the extended operation...\n" );
    if ( ( rc = ldap_extended_operation_s( ld, oidrequest, &valrequest, NULL,
NULL, &oidresult, &valresult ) ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_extended_operation failed: " );
  ldap_unbind( ld );
  return( 1 );
```

Code Example 15-1 Requesting an extended operation

```
    }
    /* Get the OID and the value from the result returned by the server. */
    printf( "Operation successful.\n" );
    printf( "\tReturned OID: %s\n", oidresult );
    printf( "\tReturned value: %s\n", valresult->bv_val );
    /* Disconnect from the server. */
    ldap_unbind( ld );
    return 0;
}
```

Chapter 16

Writing Multithreaded Clients

This chapter explains how to write multi-threaded LDAP clients.

The chapter consists of the following sections:

- Specifying Thread Functions
- Example of a Pthreads Client Application

Specifying Thread Functions

If you are writing a multi-threaded client where different threads access the same LDAP structure, you need to set up the session so that the threads do not interfere with each other.

For example, in a single-threaded LDAP client, the `LDAP` structure contains the error code of the last LDAP operation. In a multi-threaded client, you need to set up the session so that each thread can have its own error code.

The `LDAP_OPT_THREAD_FN_PTRS` session option lets you set up an `ldap_thread_fns` structure identifying the functions that are called in multi-threaded environments (for example, functions to lock and unlock critical sections of code and to get and set errors).

Because this structure lets you specify these functions, you can use the LDAP API library in different types of threading environments.

Setting Up the ldap_thread_fns Structure

If you are writing a multi-threaded client in which different threads use the same LDAP connection, you need to set up the `ldap_thread_fns` structure and identify the functions that you want to use.

The `ldap_thread_fns` structure has the following fields:

```
struct ldap_thread_fns {
  LDAP_TF_MUTEX_ALLOC_CALLBACK   *ltf_mutex_alloc;
  LDAP_TF_MUTEX_FREE_CALLBACK    *ltf_mutex_free;
  LDAP_TF_MUTEX_LOCK_CALLBACK    *ltf_mutex_lock;
  LDAP_TF_MUTEX_UNLOCK_CALLBACK  *ltf_mutex_unlock;
  LDAP_TF_GET_ERRNO_CALLBACK     *ltf_get_errno;
  LDAP_TF_SET_ERRNO_CALLBACK     *ltf_set_errno;
  LDAP_TF_GET_LDERRNO_CALLBACK   *ltf_get_lderrno;
  LDAP_TF_SET_LDERRNO_CALLBACK   *ltf_set_lderrno;
  void    *ltf_lderrno_arg;
};
```

These fields are describe in more detail below:

Table 16-1 ldap_thread_fns field descriptions

Field	Description
`*ltf_mutex_alloc`	Function pointer for allocating a mutex. This function is called by the client when needed if the function pointer is not `NULL`.
`*ltf_mutex_free`	Function pointer for freeing a mutex. This function is called by the client when needed if the function pointer is not `NULL`.
`*ltf_mutex_lock`	Function pointer for locking critical sections of code. This function is called by the client when needed if the function pointer is not `NULL`.
`*ltf_mutex_unlock`	Function pointer for unlocking critical sections of code. This function is called by the client when needed if the function pointer is not `NULL`.
`*ltf_get_errno`	Function pointer for getting the value of the `errno` variable. This function is called by the client when needed if the function pointer is not `NULL`. In a threaded environment, `errno` is typically redefined so that it has a value for each thread, rather than a global value for the entire process. This redefinition is done at compile time. Because the `libldap` library does not know what method your code and threading environment will use to get the value of `errno` for each thread, it calls this function to return the value of `errno`.

Table 16-1 ldap_thread_fns field descriptions

Field	Description
`*ltf_set_errno`	Function pointer for setting the value of the `errno` variable. This function is called by the client when needed if the function pointer is not `NULL`. In a threaded environment, `errno` is typically redefined so that it has a value for each thread, rather than a global value for the entire process. This redefinition is done at compile time. Because the `libldap` library does not know what method your code and threading environment will use to get the value of `errno` for each thread, it calls this function to set the value of `errno`.
`*ltf_get_lderrno`	Function pointer for getting error values from calls to functions in the `libldap` library. This function is called by the client when needed if the function pointer is not `NULL`. If this function pointer is not set, the `libldap` library records these errors in fields in the `LDAP` structure.
`*ltf_set_lderrno`	Function pointer for setting error values from calls to functions in the `libldap` library. This function is called by the client when needed if the function pointer is not `NULL`. If this function pointer is not set, the `libldap` library records these errors in fields in the `LDAP` structure.
`*ltf_lderrno_arg`	Additional parameter passed to the functions for getting and setting error values from calls to functions in the `libldap` library. `*ltf_get_lderrno` and `*ltf_set_lderrno` identify these functions.

Setting Up the ldap_extra_thread_fns Structure

The LDAP SDK for C provides a structure, `ldap_extra_thread_fns`, which specifies additional thread functions for locking and for semaphores. The `ldap_extra_thread_fns` structure is declared as follows:

```
struct ldap_extra_thread_fns {
  LDAP_TF_MUTEX_TRYLOCK_CALLBACK  *ltf_mutex_trylock;
  LDAP_TF_SEMA_ALLOC_CALLBACK     *ltf_sema_alloc;
  LDAP_TF_SEMA_FREE_CALLBACK      *ltf_sema_free;
  LDAP_TF_SEMA_WAIT_CALLBACK      *ltf_sema_wait;
  LDAP_TF_SEMA_POST_CALLBACK      *ltf_sema_post;
  LDAP_TF_THREADID_CALLBACK       *ltf_threadid_fn;
};
```

The LDAP SDK for C, version 4.0 and greater, supports only the `LDAP_TF_TREADID_CALLBACK *ltf_threadid_fn` function. (If any of the other extra thread callback functions are set, they will be ignored.) The thread id callback function must return an identifier that is unique to the calling thread, like `pthread_self()` does. You can use this function callback in a multi-threaded application to improve the performance of thread locking.

Providing a thread ID callback function will allow the LDAP SDK for C to use finer grained locks and potentially improve performance.

Setting the Session Options

After you set up the `ldap_thread_fns` structure, you need to associate the structure with the current session. Call the `ldap_set_option()` function and pass `LDAP_OPT_THREAD_FN_PTRS` as the value of the option parameter. Pass a pointer to the `ldap_thread_fns` structure as the value of the `optdata` parameter.

If you also set up the ldap_extra_thread_fns structure, you can associate it with the current session by calling `ldap_set_option()` function and passing `LDAP_OPT_EXTRA_THREAD_FN_PTRS` as the value of the option parameter. You must also pass a pointer to the `ldap_extra_thread_fns` structure as the value of the `optdata` parameter.

Example of Specifying Thread Functions

For example, the following section of code sets up an `ldap_thread_fns` structure for an LDAP session.

Code Example 16-1 Setting up an ldap_thread_fns structure

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>
#include <pthread.h>
#include <ldap.h>

struct ldap_thread_fns tfns;
...
  /* Set up the ldap_thread_fns structure with pointers
    to the functions that you want called */
  memset( &tfns, '\0', sizeof(struct ldap_thread_fns) );

  /* Specify the functions that you want called */

  /* Call the my_mutex_alloc() function whenever mutexes
```

Code Example 16-1 Setting up an ldap_thread_fns structure

```
    need to be allocated */
  tfns.ltf_mutex_alloc = (void *(*)(void)) my_mutex_alloc;

  /* Call the my_mutex_free() function whenever mutexes
    need to be destroyed */
  tfns.ltf_mutex_free = (void (*)(void *)) my_mutex_free;

  /* Call the pthread_mutex_lock() function whenever a
    thread needs to lock a mutex. */
  tfns.ltf_mutex_lock = (int (*)(void *)) pthread_mutex_lock;

  /* Call the pthread_mutex_unlock() function whenever a
    thread needs to unlock a mutex. */
  tfns.ltf_mutex_unlock = (int (*)(void *)) pthread_mutex_unlock;

  /* Call the get_errno() function to get the value of errno */
  tfns.ltf_get_errno = get_errno;

  /* Call the set_errno() function to set the value of errno */
  tfns.ltf_set_errno = set_errno;

  /* Call the get_ld_error() function to get error values from
    calls to functions in the libldap library */
  tfns.ltf_get_lderrno = get_ld_error;

  /* Call the set_ld_error() function to set error values for
    calls to functions in the libldap library */
  tfns.ltf_set_lderrno = set_ld_error;

  /* Don't pass any extra parameter to the functions for
    getting and setting libldap function call errors */
  tfns.ltf_lderrno_arg = NULL;
...
/* Set the session option that specifies the functions to call for
    multi-threaded clients */
if (ldap_set_option( ld, LDAP_OPT_THREAD_FN_PTRS, (void *) &tfns)!= 0) {
  ldap_perror( ld, "ldap_set_option: thread pointers" );
}
...
```

Example of a Pthreads Client Application

The following example, which uses pthreads (POSIX threads) under Solaris, is the source code for a multi-threaded client. The client connects to a specified LDAP server and creates several threads to perform multiple search and update operations simultaneously on the directory.

Code Example 16-2 Using POSIX threading under Solaris

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>
#include <pthread.h>
#include <ldap.h>
#include <synch.h>

/* Authentication and search information. */
#define NAME         "cn=Directory Manager"
#define PASSWORD     "rtfm11111"
#define BASE         "o=airius.com"
#define SCOPE        LDAP_SCOPE_SUBTREE

/* Function declarations */
static void *search_thread();
static void *modify_thread();
static void *add_thread();
static void *delete_thread();
static void set_ld_error();
static int  get_ld_error();
static void set_errno();
static int  get_errno();
static void tsd_setup();

/* Linked list of LDAPMessage structs for search results. */
typedef struct ldapmsgwrapper {
    LDAPMessage      *lmw_messagep;
    struct ldapmsgwrapper  *lmw_next;
} ldapmsgwrapper;

LDAP    *ld;
pthread_key_t  key;

main( int argc, char **argv )
{
  pthread_attr_t  attr;
  pthread_t  search_tid, search_tid2, search_tid3, search_tid4;
  pthread_t  modify_tid, add_tid, delete_tid;
  void    *status;
  struct ldap_thread_fns  tfns;
  struct ldap_extra_thread_fns extrafns;
  int rc;

  /* Check command-line syntax. */
  if ( argc != 3 ) {
    fprintf( stderr, "usage: %s <host> <port>\n", argv[0] );
    exit( 1 );
  }

  /* Create a key. */
  if ( pthread_key_create( &key, free ) != 0 ) {
    perror( "pthread_key_create" );
  }
  tsd_setup();
```

Code Example 16-2 Using POSIX threading under Solaris

```
  /* Initialize the LDAP session. */
  if ( (ld = ldap_init( argv[1], atoi( argv[2] ) )) == NULL ) {
    perror( "ldap_init" );
    exit( 1 );
  }

  /* Set the function pointers for dealing with mutexes
     and error information. */
  memset( &tfns, '\0', sizeof(struct ldap_thread_fns) );
  tfns.ltf_mutex_alloc = (void *(*)(void)) my_mutex_alloc;
  tfns.ltf_mutex_free = (void (*)(void *)) my_mutex_free;
  tfns.ltf_mutex_lock = (int (*)(void *)) pthread_mutex_lock;
  tfns.ltf_mutex_unlock = (int (*)(void *)) pthread_mutex_unlock;
  tfns.ltf_get_errno = get_errno;
  tfns.ltf_set_errno = set_errno;
  tfns.ltf_get_lderrno = get_ld_error;
  tfns.ltf_set_lderrno = set_ld_error;
  tfns.ltf_lderrno_arg = NULL;

  /* Set up this session to use those function pointers. */
  rc = ldap_set_option( ld, LDAP_OPT_THREAD_FN_PTRS, (void *) &tfns );
  if ( rc < 0 ) {
    fprintf( stderr, "ldap_set_option (LDAP_OPT_THREAD_FN_PTRS): %s\n",
ldap_err2string( rc ) );
    exit( 1 );
  }

  /* Set the function pointers for working with semaphores. */
  memset( &extrafns, '\0', sizeof(struct ldap_extra_thread_fns) );
  extrafns.ltf_mutex_trylock = (int (*)(void *)) = null;
  extrafns.ltf_sema_alloc = (void *(*)(void)) = null;
  extrafns.ltf_sema_free = (void (*)(void *)) = null;
  extrafns.ltf_sema_wait = (int (*)(void *)) = null;
  extrafns.ltf_sema_post = (int (*)(void *)) = null;
  extrafns.ltf_threadid_fn = (void * (*)(void) )pthread_self;
  /* Set up this session to use those function pointers. */
  rc = ldap_set_option( ld, LDAP_OPT_EXTRA_THREAD_FN_PTRS, (void *) &extrafns
);
  if ( rc < 0 ) {
    fprintf( stderr, "ldap_set_option (LDAP_OPT_EXTRA_THREAD_FN_PTRS): %s\n",
ldap_err2string( rc ) );
    exit( 1 );
  }

  /* Attempt to bind to the server. */
  rc = ldap_simple_bind_s( ld, NAME, PASSWORD );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    exit( 1 );
  }

  /* Initialize the attribute. */
  if ( pthread_attr_init( &attr ) != 0 ) {
```

Code Example 16-2 Using POSIX threading under Solaris

```
    perror( "pthread_attr_init" );
    exit( 1 );
  }

  /* Specify that the threads are joinable. */
  pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_JOINABLE );

  /* Create seven threads: one for adding, one for modifying,
     one for deleting, and four for searching. */
  if ( pthread_create( &search_tid, &attr, search_thread, "1" ) != 0 ) {
    perror( "pthread_create search_thread" );
    exit( 1 );
  }
  if ( pthread_create( &modify_tid, &attr, modify_thread, "2" ) != 0 ) {
    perror( "pthread_create modify_thread" );
    exit( 1 );
  }
  if ( pthread_create( &search_tid2, &attr, search_thread, "3" ) != 0 ) {
    perror( "pthread_create search_thread2" );
    exit( 1 );
  }
  if ( pthread_create( &add_tid, &attr, add_thread, "4" ) != 0 ) {
    perror( "pthread_create add_thread" );
    exit( 1 );
  }
  if ( pthread_create( &search_tid3, &attr, search_thread, "5" ) != 0 ) {
    perror( "phread_create search_thread3" );
    exit( 1 );
  }
  if ( pthread_create( &delete_tid, &attr, delete_thread, "6" ) != 0 ) {
    perror( "pthread_create delete_thread" );
    exit( 1 );
  }
  if ( pthread_create( &search_tid4, &attr, search_thread, "7" ) != 0 ) {
    perror( "pthread_create search_thread4" );
    exit( 1 );
  }

  /* Wait until these threads exit. */
  pthread_join( modify_tid, &status );
  pthread_join( add_tid, &status );
  pthread_join( delete_tid, &status );
  pthread_join( search_tid, &status );
  pthread_join( search_tid2, &status );
  pthread_join( search_tid3, &status );
  pthread_join( search_tid4, &status );
}

/* Thread for searching the directory.
   The results are not printed out. */
static void *
search_thread( char *id )
{
  LDAPMessage  *res;
  LDAPMessage  *e;
```

Code Example 16-2 Using POSIX threading under Solaris

```
  char    *a;
  char    **v;
  char    *dn;
  BerElement  *ber;
  int    i, rc, parse_rc, msgid, finished;
  int    num_entries, num_refs;
  void    *tsd;
  struct timeval  zerotime;
  zerotime.tv_sec = zerotime.tv_usec = 0L;

  printf( "Starting search_thread %s.\n", id );
  tsd_setup();
  /* Continually search the directory. */
  for ( ;; ) {
    printf( "Thread %s: Searching...\n", id );
    finished = 0;
    num_entries = 0;
    num_refs = 0;
    rc = ldap_search_ext( ld, BASE, SCOPE, "(objectclass=*)",
      NULL, 0, NULL, NULL, NULL, LDAP_NO_LIMIT, &msgid );
    if ( rc != LDAP_SUCCESS ) {
      fprintf( stderr, "Thread %s error: ldap_search: %s\n",
        id, ldap_err2string( rc ) );
      continue;
    }

    /* Iterate through the results.  In this example,
       we don't print out all the results.  (It's easier
       to see the output from the other threads this way.) */
    while ( !finished ) {
      rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
      switch ( rc ) {
      case -1:
        rc = ldap_get_lderrno( ld, NULL, NULL );
        fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
        finished = 1;
        break;
      case 0:
        break;
      /* Keep track of the number of entries found. */
      case LDAP_RES_SEARCH_ENTRY:
        num_entries++;
        break;
      /* Keep track of the number of search references. */
      case LDAP_RES_SEARCH_REFERENCE:
        num_refs++;
        break;
      case LDAP_RES_SEARCH_RESULT:
        finished = 1;
        parse_rc = ldap_parse_result( ld, res, &rc, NULL, NULL, NULL, NULL, 1
);
        if ( parse_rc != LDAP_SUCCESS ) {
          fprintf( stderr, "Thread %s error: can't parse result code.\n", id
);
```

Code Example 16-2 Using POSIX threading under Solaris

```
          break;
        } else {
          if ( rc != LDAP_SUCCESS ) {
            fprintf( stderr, "Thread %s error: ldap_search: %s\n", id,
ldap_err2string( rc ) );
          } else {
            printf( "Thread %s: Got %d results and %d references.\n", id,
num_entries, num_refs );
          }
        }
        break;
      default:
        break;
      }
    }
  }
}

/* Thread for modifying directory entries.
This thread searches for entries and randomly selects entries from
the search results for modification. */
static void *
modify_thread( char *id )
{
  LDAPMessage      *res;
  LDAPMessage      *e;
  int              i, modentry, num_entries, msgid, rc, parse_rc, finished;
  LDAPMod          mod;
  LDAPMod          *mods[2];
  char             *vals[2];
  char            *dn;
  ldapmsgwrapper  *list, *lmwp, *lastlmwp;
  struct timeval  zerotime;
  zerotime.tv_sec = zerotime.tv_usec = 0L;

  printf( "Starting modify_thread %s.\n", id );
  tsd_setup();
  rc = ldap_search_ext( ld, BASE, SCOPE, "(objectclass=*)",
    NULL, 0, NULL, NULL, NULL, LDAP_NO_LIMIT, &msgid );
  if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "Thread %s error: Modify thread: "
    "ldap_search_ext: %s\n", id, ldap_err2string( rc ) );
    exit( 1 );
  }
  list = lastlmwp = NULL;
  finished = 0;
  num_entries = 0;
  while ( !finished ) {
    rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
    switch ( rc ) {
    case -1:
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
      exit( 1 );
```

Code Example 16-2 Using POSIX threading under Solaris

```
      break;
    case 0:
      break;

    /* Keep track of the number of entries found. */
    case LDAP_RES_SEARCH_ENTRY:
      num_entries++;
      if (( lmwp = (ldapmsgwrapper *)
        malloc( sizeof( ldapmsgwrapper ))) == NULL ) {
        fprintf( stderr, "Thread %s: Modify thread: Cannot malloc\n", id );
        exit( 1 );
      }
      lmwp->lmw_messagep = res;
      lmwp->lmw_next = NULL;
      if ( lastlmwp == NULL ) {
        list = lastlmwp = lmwp;
      } else {
        lastlmwp->lmw_next = lmwp;
      }
      lastlmwp = lmwp;
      break;
    case LDAP_RES_SEARCH_REFERENCE:
      break;
    case LDAP_RES_SEARCH_RESULT:
      finished = 1;
      parse_rc = ldap_parse_result( ld, res, &rc, NULL, NULL, NULL, NULL, 1 );
      if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "Thread %s error: can't parse result code.\n", id );
        exit( 1 );
      } else {
        if ( rc != LDAP_SUCCESS ) {
          fprintf( stderr, "Thread %s error: ldap_search: %s\n", id,
ldap_err2string( rc ) );
        } else {
          printf( "Thread %s: Got %d results.\n", id, num_entries );
        }
      }
      break;
    default:
      break;
    }
  }

  /* Set up the modifications to be made. */
  mods[0] = &mod;
  mods[1] = NULL;
  vals[0] = "bar";
  vals[1] = NULL;

  /* Modify randomly selected entries. */
  for ( ;; ) {

    /* Randomly select the entries. */
    modentry = rand() % num_entries;
    for ( i = 0, lmwp = list; lmwp != NULL && i < modentry;
```

Code Example 16-2 Using POSIX threading under Solaris

```
        i++, lmwp = lmwp->lmw_next ) {
      /* Keep iterating. */
    }
    if ( lmwp == NULL ) {
      fprintf( stderr,
          "Thread %s: Modify thread could not find entry %d of %d\n",
          id, modentry, num_entries );
      continue;
    }
    e = lmwp->lmw_messagep;
    printf( "Thread %s: Modify thread picked entry %d of %d\n", id, i,
num_entries );

    /* Perform the modification. */
    dn = ldap_get_dn( ld, e );
    mod.mod_op = LDAP_MOD_REPLACE;
    mod.mod_type = "description";
    mod.mod_values = vals;
    printf( "Thread %s: Modifying (%s)\n", id, dn );
    rc = ldap_modify_ext_s( ld, dn, mods, NULL, NULL );
            if ( rc != LDAP_SUCCESS ) {
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_modify_ext_s: %s\n", ldap_err2string( rc ) );
    }
    free( dn );
  }
}

/* Thread for adding directory entries.
This thread randomly generates DNs for entries and attempts to add them to the
directory. */
static void *
add_thread( char *id )
{
  LDAPMod  mod[5];
  LDAPMod  *mods[6];
  char  dn[BUFSIZ], name[40];
  char  *cnvals[2], *snvals[2], *ocvals[3];
  int  i, rc;

  printf( "Starting add_thread %s.\n", id );
  tsd_setup();

  /* Set up the entry to be added. */
  for ( i = 0; i < 5; i++ ) {
    mods[i] = &mod[i];
  }
  mods[5] = NULL;
  mod[0].mod_op = 0;
  mod[0].mod_type = "cn";
  mod[0].mod_values = cnvals;
  cnvals[1] = NULL;
  mod[1].mod_op = 0;
  mod[1].mod_type = "sn";
```

Code Example 16-2 Using POSIX threading under Solaris

```
  mod[1].mod_values = snvals;
  snvals[1] = NULL;
  mod[2].mod_op = 0;
  mod[2].mod_type = "objectclass";
  mod[2].mod_values = ocvals;
  ocvals[0] = "top";
  ocvals[1] = "person";
  ocvals[2] = NULL;
  mods[3] = NULL;

  /* Randomly generate DNs and add entries. */
  for ( ;; ) {
    sprintf( name, "%d", rand() );
    sprintf( dn, "cn=%s, " BASE, name );
    cnvals[0] = name;
    snvals[0] = name;
    printf( "Thread %s: Adding entry (%s)\n", id, dn );
    rc = ldap_add_ext_s( ld, dn, mods, NULL, NULL );
            if ( rc != LDAP_SUCCESS ) {
      rc = ldap_get_lderrno( ld, NULL, NULL );
      fprintf( stderr, "ldap_add_ext_s: %s\n", ldap_err2string( rc ) );
    }
  }
}

/* Thread for deleting directory entries.
This thread randomly selects entries for deletion. */
static void *
delete_thread( char *id )
{
  LDAPMessage  *res;
  char    dn[BUFSIZ], name[40];

  printf( "Starting delete_thread %s.\n", id );
  tsd_setup();

  /* Randomly select entries for deletion. */
  for ( ;; ) {
    sprintf( name, "%d", rand() );
    sprintf( dn, "cn=%s, " BASE, name );
    printf( "Thread %s: Deleting entry (%s)\n", id, dn );
    if ( ldap_delete_ext_s( ld, dn, NULL, NULL ) != LDAP_SUCCESS ) {
      ldap_perror( ld, "ldap_delete_ext_s" );
    }
  }
}

/* Function for allocating a mutex. */
static void *
my_mutex_alloc( void )
{
  pthread_mutex_t  *mutexp;
  if ( (mutexp = malloc( sizeof(pthread_mutex_t) )) != NULL ) {
    pthread_mutex_init( mutexp, NULL );
  }
```

Code Example 16-2 Using POSIX threading under Solaris

```
  return( mutexp );
}

/* Function for freeing a mutex. */
static void
my_mutex_free( void *mutexp )
{
  pthread_mutex_destroy( (pthread_mutex_t *) mutexp );
  free( mutexp );
}

/* Error structure. */
struct ldap_error {
  int  le_errno;
  char  *le_matched;
  char  *le_errmsg;
};

/* Function to set up thread-specific data. */
static void
tsd_setup()
{
  void  *tsd;
  tsd = pthread_getspecific( key );
  if ( tsd != NULL ) {
    fprintf( stderr, "tsd non-null!\n" );
    pthread_exit( NULL );
  }
  tsd = (void *) calloc( 1, sizeof(struct ldap_error) );
  pthread_setspecific( key, tsd );
}

/* Function for setting an LDAP error. */
static void
set_ld_error( int err, char *matched, char *errmsg, void *dummy )
{
  struct ldap_error *le;
  le = pthread_getspecific( key );
  le->le_errno = err;
  if ( le->le_matched != NULL ) {
    ldap_memfree( le->le_matched );
  }
  le->le_matched = matched;
  if ( le->le_errmsg != NULL ) {
    ldap_memfree( le->le_errmsg );
  }
  le->le_errmsg = errmsg;
}

/* Function for getting an LDAP error. */
static int
get_ld_error( char **matched, char **errmsg, void *dummy )
{
  struct ldap_error *le;
  le = pthread_getspecific( key );
```

Code Example 16-2 Using POSIX threading under Solaris

```
  if ( matched != NULL ) {
    *matched = le->le_matched;
  }
  if ( errmsg != NULL ) {
    *errmsg = le->le_errmsg;
  }
  return( le->le_errno );
}

/* Function for setting errno. */
static void
set_errno( int err )
{
  errno = err;
}

/* Function for getting errno. */
static int
get_errno( void )
{
  return( errno );
}
```

Part 4

Reference

Chapter 17

Data Types and Structures

This chapter describes the data structures and data types used by functions in the LDAP API.

The chapter documents the following data structures and data types:

- `berval`
- `BerElement`
- `FriendlyMap`
- `LDAP`
- `LDAP_CMP_CALLBACK`
- `LDAPControl`
- `LDAP_DNSFN_GETHOSTBYADDR`
- `LDAP_DNSFN_GETHOSTBYNAME`
- `ldap_dns_fns`
- `ldap_extra_thread_fns`
- `LDAPFiltDesc`
- `LDAPFiltInfo`
- `LDAPHostEnt`
- `LDAP_IOF_CLOSE_CALLBACK`
- `LDAP_IOF_CONNECT_CALLBACK`
- `LDAP_IOF_IOCTL_CALLBACK`
- `LDAP_IOF_READ_CALLBACK`

- LDAP_IOF_SELECT_CALLBACK
- LDAP_IOF_SOCKET_CALLBACK
- LDAP_IOF_SSL_ENABLE_CALLBACK
- LDAP_IOF_WRITE_CALLBACK
- ldap_io_fns
- LDAPMemCache
- LDAPMessage
- LDAPMod
- LDAP_REBINDPROC_CALLBACK
- LDAPsortkey
- LDAP_TF_GET_ERRNO_CALLBACK
- LDAP_TF_SET_ERRNO_CALLBACK
- LDAP_TF_GET_LDERRNO_CALLBACK
- LDAP_TF_SET_LDERRNO_CALLBACK
- LDAP_TF_MUTEX_ALLOC_CALLBACK
- LDAP_TF_MUTEX_FREE_CALLBACK
- LDAP_TF_MUTEX_LOCK_CALLBACK
- LDAP_TF_MUTEX_TRYLOCK_CALLBACK
- LDAP_TF_MUTEX_UNLOCK_CALLBACK
- LDAP_TF_SEMA_ALLOC_CALLBACK
- LDAP_TF_SEMA_FREE_CALLBACK
- LDAP_TF_SEMA_POST_CALLBACK
- LDAP_TF_SEMA_WAIT_CALLBACK
- LDAP_TF_THREADID_CALLBACK
- ldap_thread_fns
- LDAPURLDesc
- LDAP_VALCMP_CALLBACK

- LDAPVersion
- LDAPVirtualList

berval

berval represents binary data that is encoded using simplified Basic Encoding Rules (BER). The data and size of the data are included in a berval structure.

berval is defined as follows:

```
struct berval {
  unsigned long bv_len;
  char *bv_val;
};
```

The fields in this structure are described below:

Table 17-1 berval field descriptions

bv_len	The length of the data.
bv_val	The binary data.

Use a berval structure when working with attributes that contain binary data (such as a JPEG or audio file).

BerElement

BerElement represents data encoded using the Basic Encoding Rules (BER). BerElement is not completely exposed in ldap.h because the fields within the structure are not intended to be accessible to clients.

Calling the ldap_first_attribute() function allocates a BerElement structure in memory. You use this structure to keep track of the current attribute. When you are done reading attributes in an entry, you need to free the BerElement structure from memory by calling the ldap_ber_free() function.

FriendlyMap

FriendlyMap represents the mapping between a list of "unfriendly names" and "friendly names". For example, you can represent the list of two-letter state codes (cryptic, "unfriendly names") and corresponding state names (easy to understand, "friendly names") in a FriendlyMap structure.

FriendlyMap is not completely defined in ldap.h because the fields within the structure are not intended to be accessible to clients.

Calling the ldap_friendly_name() routine allocates a FriendlyMap structure and reads a list of "unfriendly names" and "friendly names" from a file.

LDAP

LDAP is a type of structure representing the connection with the LDAP server. LDAP is not completely defined in ldap.h because the fields within the structure are not intended to be accessible to clients.

When you call functions that perform LDAP operations on an LDAP server (for example, when you call ldap_search_ext() to search the directory or ldap_modify_ext() to update an entry in the directory), you need to pass a pointer to an LDAP structure as a parameter to the function.

To create, manipulate, and free an LDAP structure, call the following functions:

- To create an LDAP structure, call the ldap_init() or ldapssl_init() function.
- To view or modify the properties of the connection, call the ldap_get_option() and ldap_set_option() functions.
- To close the connection and free the LDAP structure, call the ldap_unbind() or ldap_unbind_s() function.

LDAP_CMP_CALLBACK

LDAP_CMP_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and specify it when calling the ldap_sort_entries() or ldap_multisort_entries() function, your function will be called by your LDAP client. The client will call your function to sort a specified set of entries.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_CMP_CALLBACK)(const char*, const char*);
```

For information on the arguments, return values, and purpose of this function, see `ldap_sort_entries()` and `ldap_multisort_entries()`.

LDAPControl

`LDAPControl` represents a client or server control associated with an LDAP operation. Controls are part of the LDAPv3 protocol. You can use a control to extend the functionality of an LDAP operation.

There are two basic types of controls described in the LDAPv3 protocol:

- Server controls are controls that are sent from the client to the server along with an LDAP request. (In some cases, a server can include a control in the response it sends back to the client.)

  For example, you can include a server control in a search request to specify that you want the server to sort the search results before sending them back.

- Client controls are controls that can extend the client but that are never sent to the server. As a general example, you might be able to pass a client control to an LDAP API function, which might parse the control and use the data that you've specified in the control.

  Note that the Netscape LDAP SDK for C, v4.1 does not currently support any client controls.

`LDAPControl` is defined as follows:

```
typedef struct ldapcontrol {
  char *ldctl_oid;
  struct berval ldctl_value;
  char ldctl_iscritical;
} LDAPControl;
```

The fields in this structure are described below:

Table 17-2 LDAPControl field descriptions

`ldctl_oid`	Object identifier (OID) of the control.
`ldctl_value`	`berval` structure containing data associated with the control. If you want to specify a zero-length value, set `ldctl_value.bv_len` to 0 and `ldctl_value.bv_val` to a zero-length string. To indicate that no data is associated with the control, set `ldctl_value.bv_val` to `NULL`.

Table 17-2 LDAPControl field descriptions

`ldctl_iscritical`	Specifies whether or not the control is critical to the operation. This field can have one of the following values: • A non-zero value specifies that the control is critical to the operation. • `0` specifies that the control is not critical to the operation.

For more information on LDAP controls, see Chapter 14, "Working with LDAP Controls."

LDAP_DNSFN_GETHOSTBYADDR

`LDAP_DNSFN_GETHOSTBYADDR` specifies the prototype for a callback function. This callback function should be equivalent to the `gethostbyaddr_r()` function that is available on some UNIX platforms.

If you define a function with this prototype and set it in the `ldap_dns_fns` structure, your function will be called by the LDAP SDK for C on behalf of your LDAP client if it needs to get the hostname of the LDAP server to which it is connected.

The prototype specified by this data type is:

```
typedef LDAPHostEnt * (LDAP_C LDAP_CALLBACK
LDAP_DNSFN_GETHOSTBYADDR)( const char *addr, int length, int type,
  LDAPHostEnt *result, char *buffer, int buflen, int *statusp,
  void *extradata );
```

LDAP_DNSFN_GETHOSTBYNAME

`LDAP_DNSFN_GETHOSTBYNAME` specifies the prototype for a callback function. This callback function should be equivalent to the `gethostbyname_r()` function that is available on some UNIX platforms.

If you define a function with this prototype and set it in the `ldap_dns_fns` structure, your function will be called by the LDAP SDK on behalf of your LDAP client to get the host entry for the LDAP server when connecting to the server.

The prototype specified by this data type is:

```
typedef LDAPHostEnt * (LDAP_C LDAP_CALLBACK
LDAP_DNSFN_GETHOSTBYADDR)( const char *addr, int length, int type,
  LDAPHostEnt *result, char *buffer, int buflen, int *statusp,
  void *extradata );
```

ldap_dns_fns

`ldap_dns_fns` contains a set of pointers to DNS functions (equivalents to the `gethostbyname_r()` and `gethostbyaddr_r()` functions that are available on some UNIX platforms.)

You can use this if you want the LDAP SDK for C to call these functions when looking up the hostname or IP address for the LDAP server. For example, you could use this to call versions of the DNS functions that are safe for use in a multi-threaded application.

After you set the fields in this structure, you can register the functions in this structure for use by the client by calling the `ldap_set_option()` function and set the `LDAP_OPT_DNS_FN_PTRS` option to this structure.

`ldap_dns_fns` is defined as follows:

```
struct ldap_dns_fns {
  void *lddnsfn_extradata;
  int lddnsfn_bufsize;
  LDAP_DNSFN_GETHOSTBYNAME *lddnsfn_gethostbyname;
  LDAP_DNSFN_GETHOSTBYADDR *lddnsfn_gethostbyaddr;
};
```

The fields in this structure are described below:

Table 17-3 ldap_dns_fns field descriptions

`lddnsfn_extradata`	Value passed in the `extradata` argument of the `LDAP_DNSFN_GETHOSTBYADDR` and `LDAP_DNSFN_GETHOSTBYADDR` function calls.
`lddnsfn_bufsize`	Specifies the size of the buffer that you want passed to your DNS callback function. Your LDAP client passes this value as the `buflen` argument of the `LDAP_DNSFN_GETHOSTBYADDR` and `LDAP_DNSFN_GETHOSTBYADDR` function calls.

Table 17-3 ldap_dns_fns field descriptions

`lddnsfn_gethostbyname`	Function pointer for getting the host entry for the LDAP server. This function is called by the client when connecting to the server if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_DNSFN_GETHOSTBYNAME`. If `NULL`, the standard built-in OS routine is used.
`lddnsfn_gethostbyaddr`	Function pointer for getting the host name of the LDAP server. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_DNSFN_GETHOSTBYADDR`. If `NULL`, the standard built-in OS routine is used.

ldap_extra_thread_fns

The `ldap_extra_thread_fns` structure contains a set of pointers to additional functions that you want to use when write a multithreaded client. Your client calls these functions when getting results from the LDAP structure.

Note that the LDAP SDK for C ignores all the elements in this structure except for the `ltf_threadid_fn` function. Calling `ltf_threadid_fn` will, in some cases, enhance the performance of a multithreaded program.

After you set the fields in this structure, you can register the functions in this structure for use by the client by calling the `ldap_set_option()` function and set the `LDAP_OPT_EXTRA_THREAD_FN_PTRS` option to this structure.

The `ldap_extra_thread_fns` structure is defined as follows:

```
struct ldap_extra_thread_fns {
  LDAP_TF_MUTEX_TRYLOCK_CALLBACK  *ltf_mutex_trylock;
  LDAP_TF_SEMA_ALLOC_CALLBACK     *ltf_sema_alloc;
  LDAP_TF_SEMA_FREE_CALLBACK      *ltf_sema_free;
  LDAP_TF_SEMA_WAIT_CALLBACK      *ltf_sema_wait;
  LDAP_TF_SEMA_POST_CALLBACK      *ltf_sema_post;
  LDAP_TF_THREADID_CALLBACK       *ltf_threadid_fn;
};
```

The fields in this structure are described below:

Table 17-4 ldap_extra_thread_fns field descriptions

`ltf_mutex_trylock`	Function pointer for attempting to lock a mutex. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_MUTEX_TRYLOCK_CALLBACK`.
`ltf_sema_alloc`	Function pointer for allocating a semaphore. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_SEMA_ALLOC_CALLBACK`.
`ltf_sema_free`	Function pointer for freeing a semaphore. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_SEMA_FREE_CALLBACK`.
`ltf_sema_wait`	Function pointer for waiting for the value of a semaphore to be greater than 0. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_SEMA_WAIT_CALLBACK`.
`ltf_sema_post`	Function pointer for incrementing the value of a semaphore. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_SEMA_POST_CALLBACK`.
`ltf_threadid_fn`	Function pointer that is called to retieve the unique identifier for the calling thread. If this is `NULL`, it is not used. An example of a similar function in the POSIX threads standard is `pthread_self()`. The function must have the prototype specified by `LDAP_TF_THREADID_CALLBACK`.

For an example of setting up the `ldap_extra_thread_fns` structure, see Chapter 16, “Writing Multithreaded Clients.”

LDAPFiltDesc

LDAPFiltDesc is a type of structure that is returned when you call the ldap_init_getfilter() routine to load a filter configuration file. LDAPFiltDesc is not completely defined in ldap.h because the fields within the structure are not intended to be accessible to clients.

After calling the ldap_init_getfilter() routine, use the pointer to the returned LDAPFiltDesc structure in subsequent calls to get information about filters in the filter configuration file.

LDAPFiltInfo

LDAPFiltInfo represents information about a filter in the filter configuration file. When you call the ldap_getfirstfilter() or ldap_getnextfilter() routines to get a filter from the filter configuration file, the routines return a pointer to an LDAPFiltInfo structure containing the information about the filter.

LDAPFiltInfo is defined as follows:

```
typedef struct ldap_filt_info {
  char *lfi_filter;
  char *lfi_desc;
  int lfi_scope;
  int lfi_isexact;
  struct ldap_filt_info *lfi_next;
} LDAPFiltInfo;
```

The fields in this structure are described below:

Table 17-5 LDAPFilterInfo field descriptions

lfi_filter	The filter (for example, (cn=d*)).
lfi_desc	Description of the filter (the fifth field in the filter configuration file).

Table 17-5 LDAPFilterInfo field descriptions

`lfi_scope`	The scope of the filter (the sixth field in the filter configuration file), which can be one of the following values: • `LDAP_SCOPE_BASE` specifies that the search will be restricted to the current distinguished name. • `LDAP_SCOPE_ONELEVEL` specifies that the search will be restricted to the entries at the level beneath the current distinguished name. • `LDAP_SCOPE_SUBTREE` specifies that the search will encompass entries at all levels beneath the current distinguished name. If the scope of the filter is not specified in the filter configuration file, the scope is `LDAP_SCOPE_SUBTREE` by default.
`lfi_isexact`	Specifies whether or not the filter is an exact filter (an exact filter contains no wildcards and does not match words that sound alike): • `0` specifies that the filter is not an exact filter. • `1` specifies that the filter is an exact filter.
`lfi_next`	Pointer to the `LDAPFiltInfo` structure representing the next filter in the filter list.

The following section of code prints out information about a filter:

```
LDAPFiltInfo *lfip;
/* Print out the filter */
printf( "Filter:\t%s\n", lfdp->lfd_filter );
printf( "Description:\t%s\n", lfdp->lfd_desc );
```

For example, in the filter configuration file, if the first filter that applies to the value "@" is:

```
"@" " " "(mail=%v)" "email address is" "onelevel"
```

The code prints out:

```
Filter: (mail=@)
Description: email address is
```

LDAPHostEnt

LDAPHostEnt represents an entry for a host found by a domain name server. This type is similar to the hostent structure returned by functions such as gethostbyname_r() on UNIX.

If you are writing your own DNS functions for use by the client, your functions should return the host entry in this type of structure. See the documentation on LDAP_DNSFN_GETHOSTBYADDR and LDAP_DNSFN_GETHOSTBYNAME for details.

The fields in this structure should point to addresses within the buffer that is passed to the DNS callback (referenced in the LDAP callback function). This buffer contains the host data. The pointers in the hostent structure returned by the function point to the data in this buffer.

LDAPHostEnt is defined as follows:

```
typedef struct LDAPHostEnt {
  char *ldaphe_name;
  char **ldaphe_aliases;
  int ldaphe_addrtype;
  int ldaphe_length;
  char **ldaphe_addr_list;
} LDAPHostEnt;
```

The fields in this structure are described below:

Table 17-6 LDAPHostEnt field descriptions

ldaphe_name	Canonical name of the host.
ldaphe_aliases	List of aliases for this host.
ldaphe_addrtype	Address type of the host.
ldaphe_length	Length of the address.
ldaphe_addr_list	List of addresses for this host (as returned by the name server).

LDAP_IOF_CLOSE_CALLBACK

LDAP_IOF_CLOSE_CONNECT_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard close() system call.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_IOF_CLOSE_CALLBACK )( LBER_SOCKET );
```

LDAP_IOF_CONNECT_CALLBACK

LDAP_IOF_CONNECT_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard connect() network I/O function.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_IOF_CONNECT_CALLBACK )( LBER_SOCKET,
  struct sockaddr *, int );
```

LDAP_IOF_IOCTL_CALLBACK

LDAP_IOF_IOCTL_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard ioctl() system call.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_IOF_IOCTL_CALLBACK)( LBER_SOCKET, int, ... );
```

LDAP_IOF_READ_CALLBACK

LDAP_IOF_READ_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard read() I/O function.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_IOF_READ_CALLBACK)( LBER_SOCKET, void *, int );
```

LDAP_IOF_SELECT_CALLBACK

LDAP_IOF_SELECT_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard select() I/O function.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_SELECT_CALLBACK)( int, fd_set *, fd_set *,
  fd_set *, struct timeval * );
```

LDAP_IOF_SOCKET_CALLBACK

LDAP_IOF_SOCKET_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard socket() network I/O function.

The prototype specified by this data type is:

```
typedef LBER_SOCKET (LDAP_C LDAP_CALLBACK
  LDAP_IOF_SOCKET_CALLBACK)( int, int, int );
```

LDAP_IOF_SSL_ENABLE_CALLBACK

LDAP_IOF_SSL_ENABLE_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the ssl_enable() function.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_IOF_SSL_ENABLE_CALLBACK )( LBER_SOCKET );
```

LDAP_IOF_WRITE_CALLBACK

LDAP_IOF_WRITE_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_io_fns structure, your function will be called by your LDAP client.

This callback function is equivalent to the standard write() I/O function.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK LDAP_IOF_WRITE_CALLBACK)
  ( LBER_SOCKET, const void *, int );
```

ldap_io_fns

The ldap_io_fns structure contains a set of pointers to input/output functions that you want used with the Directory Server API. You need to set up this structure if you want to connect to the LDAP server using a secure sockets layer (SSL).

The ldap_io_fns structure is defined as follows:

```
struct ldap_io_fns {
  LDAP_IOF_READ_CALLBACK *liof_read;
  LDAP_IOF_WRITE_CALLBACK *liof_write;
  LDAP_IOF_SELECT_CALLBACK *liof_select;
  LDAP_IOF_SOCKET_CALLBACK *liof_socket;
  LDAP_IOF_IOCTL_CALLBACK *liof_ioctl;
  LDAP_IOF_CONNECT_CALLBACK *liof_connect;
  LDAP_IOF_CLOSE_CALLBACK *liof_close;
  LDAP_IOF_SSL_ENABLE_CALLBACK *liof_ssl_enable;
};
```

The fields in this structure are described below:

Table 17-7 ldap_io_fns field descriptions

liof_read	Function pointer to the equivalent of the standard read() I/O function. The function must have the prototype specified by LDAP_IOF_READ_CALLBACK.
liof_write	Function pointer to the equivalent of the standard write() I/O function. The function must have the prototype specified by LDAP_IOF_WRITE_CALLBACK.
liof_select	Function pointer to the equivalent of the standard select() I/O function. The function must have the prototype specified by LDAP_IOF_SELECT_CALLBACK.

Table 17-7 ldap_io_fns field descriptions

`liof_socket`	Function pointer to the equivalent of the standard `socket()` network I/O function. The function must have the prototype specified by `LDAP_IOF_SOCKET_CALLBACK`.
`liof_ioctl`	Function pointer to the equivalent of the standard `ioctl()` system call. The function must have the prototype specified by `LDAP_IOF_IOCTL_CALLBACK`.
`liof_connect`	Function pointer to the equivalent of the standard `connect()` network I/O function. The function must have the prototype specified by `LDAP_IOF_CONNECT_CALLBACK`.
`liof_close`	Function pointer to the equivalent of the standard `close()` system call. The function must have the prototype specified by `LDAP_IOF_CLOSE_CALLBACK`.
`liof_ssl_enable`	Function pointer to the equivalent of the `ssl_enable()` function. The function must have the prototype specified by `LDAP_IOF_SSL_ENABLE_CALLBACK`.

LDAPMemCache

`LDAPMemCache` is a type of structure representing an in-memory, client-side cache.

You can create a cache and specify the following information:

- The maximum size of the cache.
- The maximum amount of time to keep an item in the cache.
- A set of base DNs for the search requests that you want to cache (optional).
- A set of functions that you want used to ensure the thread-safety of the cache.

To use a cache, you need to associate it with a connection handle (and LDAP structure). Before a search request is sent to the server, the cache is checked to determine if the same request was made before. If an earlier request was cached, the search results are retrieved from the cache.

Note that the cache uses the search criteria as the key to cached items. Search requests with different criteria are cached as separate items.

For example, suppose you send a search request specifying that you just want to retrieve the `uid` attribute. Your client caches the results of that search. If you send a similar search request specifying that you want to retrieve all attributes instead of just the `uid`, the results cached from the previous search are not used.

The cache uses a combination of the following information as the key to a cached item:

- The hostname and port number of the LDAP server that you are searching.
- The DN to which you are currently authenticated.
- From the search criteria, the base DN, scope, filter, attributes to be returned, and an indication of whether to return attribute types only or attribute types and values.

Use the following functions to work with the cache:

- To create an `LDAPMemCache` structure, call the `ldap_memcache_init()` function.
- To associate an `LDAPMemCache` structure with a connection handle (an `LDAP` structure), call the `ldap_memcache_set()` function.
- To get the `LDAPMemCache` structure that is associated with a connection handle (an `LDAP` structure), call the `ldap_memcache_get()` function.
- To get the cache to proactively remove expired items, call the `ldap_memcache_update()` function.
- To remove entries from the cache, call the `ldap_memcache_flush()` function.
- To free the `LDAPMemCache` structure, call `ldap_memcache_destroy()`.

LDAPMessage

`LDAPMessage` is a type of structure representing the results of an LDAP operation, a chain of search results, an entry in the search results, or a search reference in the search results. `LDAPMessage` is not completely defined in `ldap.h` because the fields within the structure are not intended to be directly accessible to clients.

Calling the `ldap_search_ext_s()` or `ldap_search_ext()` followed by `ldap_result()` function creates an `LDAPMessage` structure to represent the chain of results of a search. Calling the `ldap_first_entry()` or `ldap_next_entry()` function creates an `LDAPMessage` structure to represent an entry in the search results. Calling `ldap_first_reference()` or `ldap_next_reference()` creates an `LDAPMessage` structure to represent a search reference in the search results.

To free the `LDAPMessage` structure, call the `ldap_msgfree()` routine.

LDAPMod

LDAPMod is a type of structure that you use to specify changes to an attribute in an directory entry. Before you call the ldap_add_ext(), ldap_add_ext_s(), ldap_modify_ext(), or ldap_modify_ext_s() routines to add or modify an entry in the directory, you need to fill LDAPMod structures with the attribute values that you intend to add or change.

LDAPMod is defined as follows:

```
typedef struct ldapmod {
  int mod_op;
  char *mod_type;
  union {
    char **modv_strvals;
    struct berval **modv_bvals;
  } mod_vals;
#define mod_values mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals
} LDAPMod;
```

The fields in this structure are described below:

Table 17-8 LDAPMod field descriptions

mod_op	The operation to be performed on the attribute and the type of data specified as the attribute values. This field can have one of the following values: • LDAP_MOD_ADD adds a value to the attribute. • LDAP_MOD_DELETE removes the value from the attribute. • LDAP_MOD_REPLACE replaces all existing values of the attribute. In addition, if you are specifying binary values in the mod_bvalues field, you should use the bitwise OR operator (|) to combine LDAP_MOD_BVALUES with the operation type. For example: `mod->mod_op = LDAP_MOD_ADD | LDAP_MOD_BVALUES` Note: If you are using the structure to add a new entry, you can specify 0 for the mod_op field (unless you are adding binary values and need to specify LDAP_MOD_BVALUES). See "Adding a New Entry" for details.
mod_type	The attribute type that you want to add, delete, or replace the values of (for example, "sn" or "telephoneNumber").
mod_values	A pointer to a NULL-terminated array of string values for the attribute.

Table 17-8 LDAPMod field descriptions

`mod_bvalues`	A pointer to a `NULL`-terminated array of berval structures for the attribute.

Note the following:

- If you specify `LDAP_MOD_DELETE` in the `mod_op` field and you remove all values in an attribute, the attribute is removed from the entry.
- If you specify `LDAP_MOD_DELETE` in the `mod_op` field and `NULL` in the `mod_values` field, the attribute is removed from the entry.
- If you specify `LDAP_MOD_REPLACE` in the `mod_op` field and `NULL` in the `mod_values` field, the attribute is removed from the entry.
- If you specify `LDAP_MOD_REPLACE` in the `mod_op` field and the attribute does not exist in the entry, the attribute is added to the entry.
- If you specify `LDAP_MOD_ADD` in the `mod_op` field and the attribute does not exist in the entry, the attribute is added to the entry.

If you've allocated memory for the structures yourself, you should free the structures when you're finished by calling the `ldap_mods_free()` function.

The following section of code sets up an `LDAPMod` structure to change the email address of a user's entry to "`bjensen@airius.com`":

Code Example 17-1 Setting up an LDAPMod structure

```
LDAP *ld;
LDAPMod attribute1;
LDAPMod *list_of_attrs[2];
char *mail_values[] = { "bjensen@airius.com", NULL };
char *dn;
...
/* Identify the entry that you want changed */
char *dn = "uid=bjensen, ou=People, o=Airius.com";

/* Specify that you want to replace the value of an attribute */
attribute1.mod_op = LDAP_MOD_REPLACE;

/* Specify that you want to change the value of the mail attribute */
attribute1.mod_type = "mail";

/* Specify the new value of the mail attribute */
attribute1.mod_values = mail_values;
```

Code Example 17-1 Setting up an LDAPMod structure

```
LDAP *ld;
/* Add the change to the list of attributes that you want changed */
list_of_attrs[0] = &attribute_change;
list_of_attrs[1] = NULL;

/* Update the entry with the change */
if ( ldap_modify_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_modify_s" );
  return( 1 );
}
...
```

LDAP_REBINDPROC_CALLBACK

`LDAP_REBINDPROC_CALLBACK` specifies the prototype for a callback function. If you define a function with this prototype and specify it when calling the `ldap_set_rebind_proc()` function, your function will be called by your LDAP client. The client will call your function to retrieve authentication information when automatically following referrals to other servers.

The prototype specified by this data type is:

```
typedef int (LDAP_CALL LDAP_CALLBACK
  LDAP_REBINDPROC_CALLBACK)( LDAP *ld, char **dnp, char **passwdp,
  int *authmethodp, int freeit, void *arg);
```

For information on the arguments, return values, and purpose of this function, see `ldap_set_rebind_proc()`.

LDAPsortkey

`LDAPsortkey` represents a server control used to specify that the server should sort the search results before sending them back to the client. Controls are part of the LDAPv3 protocol.

For example, you can use `LDAPsortkey` to specify that you want the server to sort search results by the `roomNumber` attribute.

`LDAPsortkey` is defined as follows:

```
typedef struct LDAPsortkey {
  char *sk_attrtype;
  char *sk_matchruleoid;
  int sk_reverseorder;
} LDAPsortkey;
```

The fields in this structure are described below:

Table 17-9 LDAPsorkey field descriptions

`sk_attrtype`	Name of the attribute that you want to use for sorting.
`sk_matchruleoid`	Object identifier (OID) of the matching rule that you want to use for sorting.
`sk_reverseorder`	Specifies whether or not the results are sorted in reverse order. This field can have one of the following values: • A non-zero value specifies that the results should be sorted in reverse order. • `0` specifies that the results should be sorted in normal (forward) order.

To create an array of `LDAPsortkey` structures, you can call the `ldap_create_sort_keylist()` function.

To free an array of `LDAPsortkey` structures, you can call the `ldap_free_sort_keylist()` function.

LDAP_TF_GET_ERRNO_CALLBACK

`LDAP_TF_GET_ERRNO_CALLBACK` specifies the prototype for a callback function. This function is called by your LDAP client when it needs to get the value of the `errno` variable for a thread.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_TF_GET_ERRNO_CALLBACK) ( void );
```

For more details on the `errno` variable, see `ldap_thread_fns`.

LDAP_TF_SET_ERRNO_CALLBACK

LDAP_TF_SET_ERRNO_CALLBACK specifies the prototype for a callback function. This function is called by your LDAP client when it needs to set the value of the errno variable for a thread.

The prototype specified by this data type is:

```
typedef void (LDAP_C LDAP_CALLBACK
  LDAP_TF_SET_ERRNO_CALLBACK)( int );
```

For more details on the errno variable, see ldap_thread_fns.

LDAP_TF_GET_LDERRNO_CALLBACK

LDAP_TF_GET_LDERRNO_CALLBACK specifies the prototype for a callback function. This function is called by your LDAP client when it needs to retrieve the LDAP result code for an operation.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_TF_GET_LDERRNO_CALLBACK)( char **, char **, void * );
```

If you define a function with this prototype and set it in the ldap_thread_fns structure, your callback function is called when the ldap_get_lderrno() function is called. The arguments of ldap_get_lderrno() are passed to your function, and the value returned by your function is returned by ldap_get_lderrno().

For more details on the arguments and return values that your function must use, see ldap_get_lderrno().

LDAP_TF_SET_LDERRNO_CALLBACK

LDAP_TF_SET_LDERRNO_CALLBACK specifies the prototype for a callback function. This function is called by your LDAP client when it needs to set the LDAP result code for an operation.

The prototype specified by this data type is:

```
typedef void (LDAP_C LDAP_CALLBACK
  LDAP_TF_SET_LDERRNO_CALLBACK)( int, char *, char *, void * );
```

If you define a function with this prototype and set it in the ldap_thread_fns structure, your callback function is called when the ldap_set_lderrno() function is called. The arguments of ldap_set_lderrno() are passed to your function.

For more details on the arguments that your function must use, see `ldap_set_lderrno()`.

LDAP_TF_MUTEX_ALLOC_CALLBACK

`LDAP_TF_MUTEX_ALLOC_CALLBACK` specifies the prototype for a callback function. If you define a function with this prototype and set it in the `ldap_thread_fns` structure, your function will be called by your LDAP client when it needs to allocate a mutex.

The prototype specified by this data type is:

```
typedef void *(LDAP_C LDAP_CALLBACK
  LDAP_TF_MUTEX_ALLOC_CALLBACK)( void );
```

LDAP_TF_MUTEX_FREE_CALLBACK

`LDAP_TF_MUTEX_FREE_CALLBACK` specifies the prototype for a callback function. If you define a function with this prototype and set it in the `ldap_thread_fns` structure, your function will be called by your LDAP client when it needs to free a mutex.

The prototype specified by this data type is:

```
typedef void (LDAP_C LDAP_CALLBACK
  LDAP_TF_MUTEX_FREE_CALLBACK)( void * );
```

LDAP_TF_MUTEX_LOCK_CALLBACK

`LDAP_TF_MUTEX_LOCK_CALLBACK` specifies the prototype for a callback function. If you define a function with this prototype and set it in the `ldap_thread_fns` structure, your function will be called by your LDAP client when it needs to lock a mutex.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_TF_MUTEX_LOCK_CALLBACK)( void * );
```

LDAP_TF_MUTEX_TRYLOCK_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_MUTEX_UNLOCK_CALLBACK

LDAP_TF_MUTEX_UNLOCK_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_thread_fns structure, your function will be called by your LDAP client when it needs to unlock a mutex.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_TF_MUTEX_UNLOCK_CALLBACK)( void * );
```

LDAP_TF_SEMA_ALLOC_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_SEMA_FREE_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_SEMA_POST_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_SEMA_WAIT_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_THREADID_CALLBACK

LDAP_TF_THREADID_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and set it in the ldap_thread_fns structure, your function will be called when the LDAP SDK for C needs to identify a thread. This callback function should return an identifier that is unique to the calling thread, much like the POSIX pthread_self() function does.

The prototype specified by this data type is:

```
typedef void *(LDAP_C LDAP_CALLBACK
  LDAP_TF_THREADID_CALLBACK)( void );
```

ldap_thread_fns

The `ldap_thread_fns` structure contains a set of pointers to functions that you want to use when write a multithreaded client.

The `ldap_thread_fns` structure is defined as follows:

```
struct ldap_thread_fns {
  LDAP_TF_MUTEX_ALLOC_CALLBACK *ltf_mutex_alloc;
  LDAP_TF_MUTEX_FREE_CALLBACK *ltf_mutex_free;
  LDAP_TF_MUTEX_LOCK_CALLBACK *ltf_mutex_lock;
  LDAP_TF_MUTEX_UNLOCK_CALLBACK *ltf_mutex_unlock;
  LDAP_TF_GET_ERRNO_CALLBACK *ltf_get_errno;
  LDAP_TF_SET_ERRNO_CALLBACK *ltf_set_errno;
  LDAP_TF_GET_LDERRNO_CALLBACK *ltf_get_lderrno;
  LDAP_TF_SET_LDERRNO_CALLBACK *ltf_set_lderrno;
  void *ltf_lderrno_arg;
};
```

The fields in this structure are described below:

Table 17-10 ldap_thread_fns field descriptions

`ltf_mutex_alloc`	Function pointer for allocating a mutex. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_MUTEX_ALLOC_CALLBACK`.
`ltf_mutex_free`	Function pointer for freeing a mutex. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_MUTEX_FREE_CALLBACK`.
`ltf_mutex_lock`	Function pointer for locking critical sections of code. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_MUTEX_LOCK_CALLBACK`.
`ltf_mutex_unlock`	Function pointer for unlocking critical sections of code. This function is called by the client when needed if the function pointer is not `NULL`. The function must have the prototype specified by `LDAP_TF_MUTEX_UNLOCK_CALLBACK`.

Table 17-10 ldap_thread_fns field descriptions

`ltf_get_errno`	Function pointer for getting the value of the `errno` variable. This function is called by the client when needed if the function pointer is not `NULL`. In a threaded environment, `errno` is typically redefined so that it has a value for each thread, rather than a global value for the entire process. This redefinition is done at compile time. Because the `libldap` library does not know what method your code and threading environment will use to get the value of `errno` for each thread, it calls this function to obtain the value of `errno`. The function must have the prototype specified by `LDAP_TF_GET_ERRNO_CALLBACK`.
`ltf_set_errno`	Function pointer for setting the value of the `errno` variable. This function is called by the client when needed if the function pointer is not NULL. In a threaded environment, `errno` is typically redefined so that it has a value for each thread, rather than a global value for the entire process. This redefinition is done at compile time. Because the `libldap` library does not know what method your code and threading environment will use to get the value of `errno` for each thread, it calls this function to set the value of `errno`. The function must have the prototype specified by `LDAP_TF_SET_ERRNO_CALLBACK`.
`ltf_get_lderrno`	Function pointer for getting error values from calls to functions in the `libldap` library. This function is called by the client as needed if the function pointer isn't `NULL`. If this function pointer is not set, the `libldap` library records these errors in fields in the `LDAP` structure. The function must have the prototype specified by `LDAP_TF_GET_LDERRNO_CALLBACK`.
`ltf_set_lderrno`	Function pointer for setting error values from calls to functions in the `libldap` library. This function is called by the client as needed if the function pointer isn't `NULL`. If this function pointer is not set, the `libldap` library records these errors in fields in the `LDAP` structure. The function must have the prototype specified by `LDAP_TF_SET_LDERRNO_CALLBACK`.

Table 17-10 ldap_thread_fns field descriptions

`ltf_lderrno_arg`	Additional parameter passed to the functions for getting and setting error values from calls to functions in the `libldap` library. (`*ltf_get_lderrno`) and (`*ltf_set_lderrno`) identify these functions.

For an example of setting up the `ldap_thread_fns` structure, see Chapter 16, "Writing Multithreaded Clients."

LDAPURLDesc

`LDAPURLDesc` is a type of structure that represents the components of an LDAP URL. LDAP URLs have the following syntax:

`ldap://`*`hostport/dn`*`[?`*`attributes`*`[?`*`scope`*`[?`*`filter`*`]]]`

For example:

`ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich`

Calling the `ldap_url_parse()` routine creates an `LDAPURLDesc` structure containing the components of the URL. To free the `LDAPURLDesc` structure, call the `ldap_free_urldesc()` routine.

`LDAPURLDesc` is defined as follows:

```
typedef struct ldap_url_desc {
  char *lud_host;
  int lud_port;
  char *lud_dn;
  char **lud_attrs;
  int lud_scope;
  char *lud_filter;
  unsigned long lud_options;
} LDAPURLDesc;
```

The fields in this structure are described below:

Table 17-11 LDAPURLDesc field descriptions

`lud_host`	Name of the host in the URL.
`lud_port`	Number of the port in the URL.
`lud_dn`	Distinguished name in the URL. This "base entry" DN identifies the starting point of the search.

Table 17-11 LDAPURLDesc field descriptions

`lud_attrs`	Pointer to a NULL-terminated list of the attributes specified in the URL.
`lud_scope`	Integer representing the scope of the search specified in the URL: • `LDAP_SCOPE_BASE` specifies a search of the base entry. • `LDAP_SCOPE_ONELEVEL` specifies a search of all entries one level under the base entry (not including the base entry). • `LDAP_SCOPE_SUBTREE` specified a search of all entries at all levels under the base entry (including the base entry).
`lud_filter`	Search filter included in the URL.
`lud_options`	Options (if `LDAP_URL_OPT_SECURE`, indicates that the protocol is `ldaps://` instead of `ldap://`).

For example, suppose you pass the following URL to the `ldap_url_parse()` function:

```
ldap://ldap.iplanet.com:5000/dc=airius,dc=com?cn,mail, \
  telephoneNumber?sub?(sn=Jensen)
```

The resulting `LDAPURLDesc` structure (`ludpp`, in this example) will contain the following values:

Table 17-12 Using LDAPURLDesc to parse a URL

`ludpp->lud_host`	`ldap.iplanet.com`
`ludpp->lud_port`	`5000`
`ludpp->lud_dn`	`o=airius.com`
`ludpp->lud_attrs[0]`	`cn`
`ludpp->lud_attrs[1]`	`mail`
`ludpp->lud_attrs[2]`	`telephoneNumber`
`ludpp->lud_attrs[3]`	`NULL`
`ludpp->lud_scope`	`LDAP_SCOPE_SUBTREE`
`ludpp->lud_filter`	`(sn=Jensen)`

LDAP_VALCMP_CALLBACK

LDAP_VALCMP_CALLBACK specifies the prototype for a callback function. If you define a function with this prototype and specify it when calling the ldap_sort_values() function, your function will be called by your LDAP client. The client will call your function to sort a specified set of values.

The prototype specified by this data type is:

```
typedef int (LDAP_C LDAP_CALLBACK
  LDAP_VALCMP_CALLBACK)(const char**, const char**);
```

For information on the arguments, return values, and purpose of this function, see ldap_sort_values().

LDAPVersion

LDAPVersion is deprecated; you should use the function ldap_get_option() in its place (it is documented here for backward compatibility only).

LDAPVersion contains version information about the LDAP SDK for C. Call the ldap_version() function to return a pointer to an LDAPVersion structure containing the version information.

LDAPVersion is defined as follows:

```
typedef struct _LDAPVersion {
  int sdk_version;
  int protocol_version;
  int SSL_version;
  int security_level;
} LDAPVersion;
```

The fields in this structure are described below:

Table 17-13 LDAPVersion field descriptions

sdk_version	Version number of the LDAP SDK for C multiplied by 100 (for example, the value 100 in this field represents version 1.0).
protocol_version	Highest supported LDAP protocol version multiplied by 100 (for example, the value 300 in this field represents LDAPv3).
SSL_version	Supported SSL version multiplied by 100 (for example, the value 300 in this field represents SSL 3.0).
security_level	Level of encryption supported in bits (for example, 128 for domestic or 40 for export). If SSL is not enabled, the value of this field is LDAP_SECURITY_NONE.

LDAPVirtualList

`LDAPVirtualList` specifies the information that can be used to create a "virtual list view" control. This LDAPv3 control is designed to allow the client to retrieve subsets of search results to display in a "virtual list box".

This control is OID 2.16.840.1.113730.3.4.9, or `LDAP_CONTROL_VLVREQUEST` as defined in `ldap.h`.

This control is supported by the Netscape Directory Server, version 4.0 and later; and all iPlanet Directory Server versions. For information on determining if a server supports this or other LDAPv3 controls, see “Determining If the Server Supports LDAPv3,” on page 211.

A virtual list box is typically a graphical user interface that displays a long list of entries with a few entries visible. End users can display different sections of the list by scrolling up or down.

To display the list of entries, the client usually does not retrieve the entire list of entries from the server. Instead, the client just retrieves the subset of entries to be displayed to the end user.

The virtual list view control provides the means for your client to request and retrieve certain subsets of a long, sorted list of entries. The control specifies the following information:

- The entry in the list that is currently selected.
- The number of entries to be displayed in the list before the selected item.
- The number of entries to be displayed in the list after the selected entry.

The currently selected entry can be identified in one of the following ways:

- By the index of the entry in the entire list (in which case, the control specifies both the offset of the entry and the total number of entries in the list).
- By the value of the entry (in which case, the control specifies that value).

For example, a virtual list view control might specify that you want to retrieve entries 15 through 24 in a list of 100 results with entry 20 being the selected entry. The control uses the following information to specify this:

- The selected entry is the 20th entry from the top (in other words, the index or offset of the entry is 20) of a list of 100.
- Get 5 entries before the selected entry in the list (entries 15 - 19).
- Get 4 entries after the selected entry in the list (entries 21 - 24).

As another example, a virtual list view control might specify that you want to retrieve a subset of entries that start with the letter "c" or a later letter in the alphabet. The control might specify the following information:

- The selected entry is the first entry that starts with the letter "c". (The size of the list is not relevant in determining the selected entry in this case.)
- Get 5 entries before the selected entry in the list.
- Get 4 entries after the selected entry in the list.

`LDAPVirtualList` is defined as follows:

```
typedef struct ldapvirtuallist {
  unsigned long ldvlist_before_count;
  unsigned long ldvlist_after_count;
  char *ldvlist_attrvalue;
  unsigned long ldvlist_index;
  unsigned long ldvlist_size;
  void *ldvlist_extradata;
} LDAPVirtualList;
```

The fields in this structure are described below:

Table 17-14 LDAPVirtualList field descriptions

`ldvlist_before_count`	Number of entries before the selected entry that you want to retrieve.
`ldvlist_after_count`	Number of entries after the selected entry that you want to retrieve.
`ldvlist_attrvalue`	Specifies the value that you want to find in the list. The selected entry in the list is the first entry that is greater than or equal to this value. If this field is `NULL`, the `ldvlist_index` and `ldvlist_size` fields are used to determine the selected entry instead.
`ldvlist_index`	If the `ldvlist_attrvalue` field is `NULL`, specifies the offset or index of the selected entry in the list. This field is used in conjunction with the `ldvlist_size` field to identify the selected entry.
`ldvlist_size`	If the `ldvlist_attrvalue` field is `NULL`, specifies the total number items in the list. This field is used in conjunction with the `ldvlist_index` field to identify the selected entry.
`ldvlist_extradata`	Reserved for application-specific use. Note that this data is not used in the virtual list view control.

After you create an `LDAPVirtualList` structure and specify values for its fields, you can create the virtual list view control by calling the function `ldap_create_virtuallist_control()`.

You can pass this control and a server-side sorting control (created by calling the `ldap_create_sort_keylist()` function and the `ldap_create_sort_control()` function) to the `ldap_search_ext()` or `ldap_search_ext_s()` function.

To get the virtual list view response control sent back from the server, call the `ldap_parse_result()` function to get the list of controls returned by the server, then call the `ldap_parse_virtuallist_control()` function to retrieve information from the control.

For more information about this control, see “Using the Virtual List View Control.”

Chapter 18

Function Reference

This chapter contains a reference to the public functions of the LDAP SDK for C. Along with a detailed description of each function, the function reference details the function header file and syntax, the function parameters, and what the function returns. In many cases an example program is included with the description.

The beginning of this chapter lists the functions in the following two formats:

- "Functions (in alphabetical order)."
- "Summary of Functions by Task."

Functions (in alphabetical order)

The LDAP SDK for C includes the following functions (functions that require LDAPv3 support are noted):

- `ber_bvfree()`
- `ber_free()`
- `ldap_abandon()`
- `ldap_abandon_ext()` - LDAPv3 function
- `ldap_add()`
- `ldap_add_ext()` - LDAPv3 function
- `ldap_add_ext_s()` - LDAPv3 function
- `ldap_add_s()`
- `ldap_build_filter()`
- `ldap_compare()`

- ldap_compare_ext() - LDAPv3 function
- ldap_compare_ext_s() - LDAPv3 function
- ldap_compare_s()
- ldap_control_free() - LDAPv3 function
- ldap_controls_free() - LDAPv3 function
- ldap_count_entries()
- ldap_count_messages() - LDAPv3 function
- ldap_count_references() - LDAPv3 function
- ldap_count_values()
- ldap_count_values_len()
- ldap_create_filter()
- ldap_create_persistentsearch_control() - LDAPv3 function
- ldap_create_proxyauth_control() - LDAPv3 function
- ldap_create_sort_control() - LDAPv3 function
- ldap_create_sort_keylist() - LDAPv3 function
- ldap_create_virtuallist_control() - LDAPv3 function
- ldap_delete()
- ldap_delete_ext() - LDAPv3 function
- ldap_delete_ext_s() - LDAPv3 function
- ldap_delete_s()
- ldap_dn2ufn()
- ldap_err2string()
- ldap_explode_dn()
- ldap_explode_rdn()
- ldap_extended_operation() - LDAPv3 function
- ldap_extended_operation_s() - LDAPv3 function
- ldap_first_attribute()

- ldap_first_entry()
- ldap_first_message() - LDAPv3 function
- ldap_first_reference() - LDAPv3 function
- ldap_free_friendlymap()
- ldap_free_sort_keylist()
- ldap_free_urldesc()
- ldap_friendly_name()
- ldap_get_dn()
- ldap_get_entry_controls() - LDAPv3 function
- ldap_getfilter_free()
- ldap_getfirstfilter()
- ldap_get_lang_values()
- ldap_get_lang_values_len()
- ldap_get_lderrno()
- ldap_getnextfilter()
- ldap_get_option()
- ldap_get_values()
- ldap_get_values_len()
- ldap_init()
- ldap_init_getfilter()
- ldap_init_getfilter_buf()
- ldap_is_ldap_url()
- ldap_memcache_destroy()
- ldap_memcache_flush()
- ldap_memcache_get()
- ldap_memcache_init()
- ldap_memcache_set()

- ldap_memcache_update()
- ldap_memfree()
- ldap_modify()
- ldap_modify_ext() - LDAPv3 function
- ldap_modify_ext_s() - LDAPv3 function
- ldap_modify_s()
- ldap_modrdn()
- ldap_modrdn_s()
- ldap_modrdn2()
- ldap_modrdn2_s()
- ldap_mods_free()
- ldap_msgfree()
- ldap_msgid()
- ldap_msgtype()
- ldap_multisort_entries()
- ldap_next_attribute()
- ldap_next_entry()
- ldap_next_message() - LDAPv3 function
- ldap_next_reference() - LDAPv3 function
- ldap_parse_entrychange_control() - LDAPv3 function
- ldap_parse_extended_result() - LDAPv3 function
- ldap_parse_reference() - LDAPv3 function
- ldap_parse_result() - LDAPv3 function
- ldap_parse_sasl_bind_result() - LDAPv3 function
- ldap_parse_sort_control() - LDAPv3 function
- ldap_parse_virtuallist_control() - LDAPv3 function
- ldap_perror()

- ldap_rename() - LDAPv3 function
- ldap_rename_s() - LDAPv3 function
- ldap_result()
- ldap_result2error()
- ldap_sasl_bind() - LDAPv3 function
- ldap_sasl_bind_s() - LDAPv3 function
- ldap_search()
- ldap_search_ext() - LDAPv3 function
- ldap_search_ext_s() - LDAPv3 function
- ldap_search_s()
- ldap_search_st()
- ldap_set_filter_additions()
- ldap_setfilteraffixes()
- ldap_set_lderrno()
- ldap_set_option()
- ldap_set_rebind_proc()
- ldap_simple_bind()
- ldap_simple_bind_s()
- ldap_sort_entries()
- ldap_sort_values()
- ldap_sort_strcasecmp()
- ldap_unbind()
- ldap_unbind_s()
- ldap_unbind_ext()
- ldap_url_parse()
- ldap_url_search()
- ldap_url_search_s()

- `ldap_url_search_st()`
- `ldap_value_free()`
- `ldap_value_free_len()`
- `ldap_version()`
- `ldapssl_advclientauth_init()`
- `ldapssl_client_init()`
- `ldapssl_clientauth_init()`
- `ldapssl_enable_clientauth()`
- `ldapssl_err2string()`
- `ldapssl_init()`
- `ldapssl_install_routines()`
- `ldapssl_pkcs_init()`

Summary of Functions by Task

This section summarizes the functions in the LDAP SDK for C into the following task categories:

- Initializing and Ending LDAP Sessions
- Authenticating to an LDAP Server
- Performing LDAP Operations
- Getting Search Results
- Sorting Search Results
- Working with Search Filters
- Working with Distinguished Names
- Working with LDAPv3 Controls
- Working with LDAP URLs
- Getting the Attribute Values for a Particular Language
- Handling Errors

- Freeing Memory

Initializing and Ending LDAP Sessions

Call the following functions to initialize a session, set session options, and end a session.

Table 18-1 Functions to initialize and end an LDAP session

Function	Description
`ldap_init()`	Initialize an LDAP session.
`ldapssl_init()`	Initialize an LDAP session over SSL.
`ldapssl_pkcs_init()`	Initialize a thread-safe session over SSL.
`ldap_set_option()`	Set session preferences.
`ldap_get_option()`	Get session preferences.
`ldap_unbind()`, `ldap_unbind_s()`, or `ldap_unbind_ext()`	End an LDAP session.

Authenticating to an LDAP Server

Call the following functions to authenticate to an LDAP server.

Table 18-2 Functions to authenticate to an LDAP server

Function	Description
`ldap_simple_bind()` or `ldap_simple_bind_s()`	Authenticate to an LDAP server using a password.
`ldap_sasl_bind()` and `ldap_parse_sasl_bind_result()`, or `ldap_sasl_bind_s()`	Authenticate to an LDAP server using a SASL mechanism.
`ldap_set_rebind_proc()`	Specify the function used to get authentication information when following referrals.

Performing LDAP Operations

Call the following functions to perform LDAP operations on a server.

Table 18-3 Functions to perform operations on an LDAP server

Function	Description
`ldap_add_ext()` or `ldap_add_ext_s()`	Add a new entry to the directory.
`ldap_modify_ext()` or `ldap_modify_ext_s()`	Modify an entry in the directory.
`ldap_delete_ext()` or `ldap_delete_ext_s()`	Delete an entry from the directory.
`ldap_rename()` or `ldap_rename_s()`	Rename an entry in the directory.
`ldap_search_ext()` or `ldap_search_ext_s()`	Search the directory.
`ldap_compare_ext()` or `ldap_compare_ext_s()`	Compare entries in the directory.
`ldap_extended_operation()` or `ldap_extended_operation_s()`	Perform an LDAPv3 extended operation.
`ldap_result()`	Check the results of an asynchronous operation.
`ldap_parse_extended_result()`	Parse the results of an LDAPv3 extended operation.
`ldap_msgfree()`	Free the results from memory.
`ldap_abandon_ext()`	Cancel an asynchronous operation.

Getting Search Results

Call the following functions to retrieve search results.

Table 18-4 Functions to search entries on an LDAP server

Function	Description
`ldap_first_message()`	Get the first message (an entry or search reference) in a chain of search results.
`ldap_next_message()`	Get the next message (an entry or search reference) in a chain of search results.

Table 18-4 Functions to search entries on an LDAP server

Function	Description
`ldap_count_messages()`	Count the number of messages (entries and search references) in a chain of search results.
`ldap_first_entry()`	Get the first entry in a chain of search results.
`ldap_next_entry()`	Get the next entry in a chain of search results.
`ldap_count_entries()`	Count the number of entries in a chain of search results.
`ldap_first_reference()`	Get the first search reference in a chain of search results.
`ldap_next_reference()`	Get the next search reference in a chain of search results.
`ldap_count_references()`	Count the number of search references in a chain of search results.
`ldap_get_dn()`	Get the distinguished name for an entry.
`ldap_first_attribute()`	Get the name of the first attribute in an entry.
`ldap_next_attribute()`	Get the name of the next attribute in an entry.
`ldap_get_values()`	Get the string values of an attribute.
`ldap_get_values_len()`	Get the binary values of an attribute.
`ldap_count_values()`	Count the string values of an attribute.
`ldap_count_values_len()`	Count the binary values of an attribute.
`ldap_get_lang_values()`	Get the string values of the specified language subtype of an attribute.
`ldap_get_lang_values_len()`	Get the binary values of the specified language subtype of an attribute.
`ldap_value_free()`	Free the memory allocated for the string values of an attribute.
`ldap_value_free_len()`	Free the memory allocated for the binary values of an attribute.

Sorting Search Results

Call the following functions to sort search results.

Table 18-5 Functions that sort search results

Function	Description
`ldap_sort_entries()`	Have your client sort entries by distinguished name or by a single attribute.
`ldap_multisort_entries()`	Have your client sort entries by multiple attributes.
`ldap_create_sort_keylist()`, `ldap_create_sort_control()`, `ldap_parse_sort_control()`	Request that the server sort the search results before sending them to your client.
`ldap_sort_values()`	Sort the values of an attribute
`ldap_sort_strcasecmp()`	A case-insensitive comparison function that you can pass to `ldap_sort_values()`.

Working with Search Filters

Call the following functions to initialize, retrieve, and build filters.

Table 18-6 Functions to initialize, retrieve, and build filters

Function	Description
`ldap_init_getfilter()`	Read a filter configuration file into memory.
`ldap_init_getfilter_buf()`	Read a filter configuration from a buffer.
`ldap_set_filter_additions()`	Specify the prefix and suffix to be added to all filters retrieved from the filter configuration.
`ldap_getfirstfilter()`	Retrieve the first matching filter from the filter configuration.
`ldap_getnextfilter()`	Retrieve the next matching filter from the filter configuration.
`ldap_getfilter_free()`	Free the filter configuration from memory.
`ldap_create_filter()`	Build a filter.

Working with Distinguished Names

Call the following functions to retrieve a distinguished name from an entry and to split a distinguished name into its component parts.

Table 18-7 Functions to retrieve distinguished names

Function	Description
`ldap_get_dn()`	Get the distinguished name for an entry.
`ldap_explode_dn()`	Split up a distinguished name into its components.
`ldap_explode_rdn()`	Split up a relative distinguished name into its components.

Working with LDAPv3 Controls

Call the following functions to work with LDAPv3 controls.

Table 18-8 Functions to work with LDAPv3 controls

Function	Description
`ldap_create_persistentsearch_control()`	Create a “persistent search” control to track changes in directory entries.
`ldap_create_sort_keylist()`, `ldap_create_sort_control()`	Create a “sorting” control to return sorted search results from the LDAP server.
`ldap_create_proxyauth_control()`	Create a “proxy authorization” control to allow an entry to act as a proxy for an alternate entry.
`ldap_add_ext()`, `ldap_add_ext_s()`, `ldap_compare_ext()`, `ldap_compare_ext_s()`, `ldap_delete_ext()`, `ldap_delete_ext_s()`, `ldap_extended_operation()`, `ldap_extended_operation_s()`, `ldap_modify_ext()`, `ldap_modify_ext_s()`, `ldap_rename()`, `ldap_rename_s()`, `ldap_sasl_bind()`, `ldap_sasl_bind_s()`, `ldap_search_ext()`, `ldap_search_ext_s()`, `ldap_abandon_ext()`	Pass LDAP controls to the server.

Table 18-8 Functions to work with LDAPv3 controls

Function	Description
`ldap_parse_result()`	Parse LDAP server controls from results sent from the server.
`ldap_get_entry_controls(), ldap_parse_entrychange_control()`	Parse an "entry change notification" control from an entry and retrieve information from the control.
`ldap_parse_sort_control()`	Parse "sorting" controls from results sent from the server.
`ldap_control_free()`	Free the memory allocated for an `LDAPControl` structure.
`ldap_controls_free()`	Free the memory allocated for an array of `LDAPControl` structures.
`ldap_unbind_ext()`	Lets you specifically name a server or client control when unbinding from the server.

Working with LDAP URLs

Call the following functions to interpret LDAP URLs.

Table 18-9 Functions to interpret LDAP URLs

Function	Description
`ldap_is_ldap_url()`	Determine if a URL is an LDAP URL.
`ldap_url_parse()`	Split up an LDAP URL into its components.
`ldap_url_search(), ldap_url_search_s(), or ldap_url_search_st()`	Perform the search specified by an LDAP URL.
`ldap_free_urldesc()`	Free the memory allocated for a parsed URL.

Getting the Attribute Values for a Particular Language

Call the following functions to get the values from a particular language subtype in an attribute.

Table 18-10 Functions to get language subtypes

Function	Description
`ldap_get_lang_values() or ldap_get_lang_values_len()`	Get an attribute's value in a particular language.

Handling Errors

Call the following functions to handle errors returned by the LDAP API functions.

Table 18-11 Functions for error handling

Function	Description
`ldap_parse_result()`	Get the error code resulting from an asynchronous LDAP operation.
`ldap_get_lderrno()`	Get information about the last error that occurred.
`ldap_set_lderrno()`	Set information about an error.
`ldap_err2string()`	Get the error message for a specific error code.
`ldapssl_err2string()`	Get the error message for a specific SSL error code.

Freeing Memory

Call the following functions to free memory allocated by the LDAP API functions.

Table 18-12 Functions to free memory

Function	Description
`ldap_memfree()`	Free memory allocated by an LDAP API function call.
`ldap_mods_free()`	Free the structures allocated for adding or modifying entries in the directory.

Table 18-12 Functions to free memory

Function	Description
`ldap_msgfree()`	Free the memory allocated for search results or other LDAP operation results.
`ldap_value_free()`	Free the memory allocated for the string values of an attribute.
`ldap_value_free_len()`	Free the memory allocated for the binary values of an attribute (an array of `berval` structures).
`ber_bvfree()`	Free the memory allocated for a `berval` structures.
`ldap_getfilter_free()`	Free the filter configuration from memory.
`ldap_free_urldesc()`	Free the memory allocated for a parsed URL.
`ber_free()`	Free the memory allocated for a `BerElement` structure.
`ldap_control_free()`	Free the memory allocated for an `LDAPControl` structure
`ldap_controls_free()`	Free the memory allocated for an array of `LDAPControl` structures.
`ldap_free_sort_keylist()`	Free the memory allocated for an array of `LDAPsortkey` structures.

ber_bvfree()

Frees a berval structure from memory.

Syntax

```
#include <lber.h>
void ber_bvfree( struct berval *bv );
```

Parameters

This function has the following parameters:

Table 18-13 ber_bvfree() function parameters

`bv`	Pointer to the `berval` structure that you want to free from memory.

Description

The `ber_bvfree()` function frees a berval structure from memory. Call this function to free berval arguments passed back from the `ldap_extended_operation_s()`, `ldap_parse_extended_result()`, `ldap_sasl_bind_s()`, and `ldap_parse_sasl_bind_result()` functions.

See Also

`ldap_extended_operation_s()`, `ldap_parse_extended_result()`, `ldap_sasl_bind_s()`, `ldap_parse_sasl_bind_result()`.

ber_free()

The `ber_free()` function frees a `BerElement` structure from memory. Call this function to free any `BerElement` structures that you have allocated.

Syntax

```
#include <ldap.h>
void ber_free( BerElement *ber, int freebuf );
```

Parameters

This function has the following parameters:

Table 18-14 ber_free() function parameters

`ber`	Pointer to the `BerElement` structure that you want to free.
`freebuf`	Specifies whether or not to free the buffer in the `BerElement` structure.

Description

You can call this function to free `BerElement` structures allocated by `ldap_first_attribute()` function calls and by `ldap_next_attribute()` function calls.

When freeing structures allocated by these functions, you should specify `0` for the `freebuf` argument. (These functions do not allocate the extra buffer in the `BerElement` structure.)

For example, to retrieve attributes from a search result entry, you need to call the `ldap_first_attribute()` function. Calling this function allocates a `BerElement` structure, which is used to keep track of the current attribute. When you are done working with the attributes, you should free this structure from memory if the structure still exists.

Example

The following example frees the `BerElement` structure allocated by the `ldap_first_attribute()` function.

Code Example 18-1 ber_free() code example

```
LDAP *ld;
LDAPMessage *a, *e;
BerElement *ber;
...
for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
  a =ldap_next_attribute( ld, e, ber ) {
  ...
  /* Retrieve the value of each attribute */
  ...
}

/* Free the BerElement when done */
if ( ber != NULL ) {
  ber_free( ber, 0 );
}
...
```

See Also

`ldap_first_attribute()`, `ldap_next_attribute()`.

ldap_abandon()

Cancels ("abandons") an asynchronous LDAP operation that is in progress.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_abandon_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_abandon( LDAP *ld, int msgid );
```

Parameters

This function has the following parameters:

Table 18-15 ldap_abandon() function parameters

`ld`	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 18-15 ldap_abandon() function parameters

`msgid`	Message ID of an LDAP operation.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- -1 if unsuccessful. The appropriate LDAP error code is also set in the `LDAP` structure. You can retrieve the error code by calling the `ldap_get_lderrno()` function.

  Some of the possible LDAP result codes for this function include:

  - `LDAP_PARAM_ERROR` (if any of the arguments are invalid).
  - `LDAP_ENCODING_ERROR` (if an error occurred when BER-encoding the request).
  - `LDAP_SERVER_DOWN` (if the LDAP server did not receive the request or if the connection to the server was lost).
  - `LDAP_NO_MEMORY` (if memory cannot be allocated).

Description

The `ldap_abandon()` function cancels ("abandons") an asynchronous LDAP operation that is in progress.

A newer version of this function, `ldap_abandon_ext()`, is available in this release of the LDAP API. `ldap_abandon()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_abandon_ext()` instead of `ldap_abandon()`.

If you want more information on `ldap_abandon()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following example cancels an `ldap_url_search()` operation, abandoning the results of the operation.

Code Example 18-2 Canceling an ldap_url_search() operation

```
LDAP *ld;
char *url = "ldap://ldap.itd.umich.edu/c=US?o,description? one?o=umich";
int msgid;
```

Code Example 18-2 Canceling an ldap_url_search() operation

```
...
/* Initiate a search operation */
msgid = ldap_url_search( ld, url, 0 );
...
/* Abandon the search operation */
if ( ldap_abandon( ld, msgid ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_abandon" );
  return( 1 );
}
...
```

See Also

`ldap_abandon_ext()`.

ldap_abandon_ext()

Cancels ("abandons") an asynchronous LDAP operation that is in progress. For example, you can cancel an LDAP search operation that you started with `ldap_search_ext()`.

Syntax

```
#include <ldap.h>
int ldap_abandon_ext( LDAP *ld, int msgid,
  LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-16 ldap_abandon_ext() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`msgid`	Message ID of the LDAP operation that you want to cancel.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.

Description

The `ldap_abandon()` function cancels ("abandons") an asynchronous LDAP operation that is in progress. For example, if you called `ldap_search_ext()` to initiate an LDAP search operation on the server, you can call `ldap_abandon_ext()` to cancel the LDAP search operation.

This function is a new version of the `ldap_abandon()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_abandon()`.

When you call this function, your LDAP client sends a request to cancel an operation being processed by the LDAP server. To identify the operation to be cancelled, specify the message ID of the operation in the `msgid` argument.

(When you call an asynchronous function such as `ldap_search_ext()` and `ldap_modify_ext()`, the `msgidp` argument of the function returns a pointer to a message ID that identifies the operation. For example, when you call `ldap_search_ext()` to start an LDAP search operation on the server, the `msgidp` argument returns a pointer to a message ID identifying that LDAP search operation.)

When you call `ldap_abandon_ext()`, the function checks to see if the results of the operation have already been returned. If so, `ldap_abandon_ext()` deletes the message ID from the queue of pending messages. If the results have not been returned, `ldap_abandon_ext()` sends a request to abandon the operation on the LDAP server.

Once you cancel an operation, results of the operation will not be returned, even if you subsequently call `ldap_result()` to try to get the results.

For more information, see "Canceling an Operation in Progress."

Example

The following example cancels an `ldap_url_search()` operation, abandoning the results of the operation.

Code Example 18-3 ldap_abandon_ext() code example

```
LDAP *ld;
char *url = "ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich";
int msgid;
LDAPControl **srvrctrls, **clntctrls;
...
/* Initiate a search operation */
msgid = ldap_url_search( ld, url, 0 );
...
/* Abandon the search operation */
if ( ldap_abandon_ext( ld, msgid, srvrctrls, clntctrls )
    != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_abandon" );
  return( 1 );
}
...
```

See Also

`ldap_add_ext()`, `ldap_compare_ext()`, `ldap_delete_ext()`, `ldap_extended_operation()`, `ldap_modify_ext()`, `ldap_rename()`, `ldap_sasl_bind()`, `ldap_search_ext()`, `ldap_simple_bind()`, `ldap_url_search()`.

ldap_add()

Adds a new entry to the directory asynchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_add_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_add( LDAP *ld, const char *dn, LDAPMod **attrs );
```

Parameters

This function has the following parameters:

Table 18-17 ldap_add() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.

Table 18-17 ldap_add() function parameters

dn	Distinguished name (DN) of the entry to add. With the exception of the leftmost component, all components of the distinguished name (for example, o=*organization* or c=*country*) must already exist.
attrs	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes of the new entry.

Returns

The message ID of the ldap_add() operation. To check the result of this operation, call ldap_result() and ldap_result2error(). See the result code documentation for the ldap_add_ext_s() function for a list of possible result codes for the LDAP add operation.

Description

The ldap_add() function adds a new entry to the directory asynchronously.

A newer version of this function, ldap_add_ext(), is available in this release of the LDAP API. ldap_add() (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use ldap_add_ext() instead of ldap_add().

If you want more information on ldap_add(), refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following example adds a new entry to the directory.

Code Example 18-4 ldap_add() code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the new entry. Note that "o=airius.com" and
"ou=People, o=airius.com" must already exist in the directory. */
char *dn = "uid=bjensen, ou=People, o=airius.com";

/* To add a "person" entry, you must specify values for the sn, cn, and
objectClass attributes. (These are required attributes.) */
```

Code Example 18-4 ldap_add() code example

```
#include <ldap.h>
char *sn_values[] = { "Jensen", NULL };

/* To specify multiple values for an attribute, add the different values to the
array. */
char *cn_values[] = { "Barbara Jensen", "Babs Jensen", NULL };

/* The object class for a "person" entry is "inetOrgPerson", which is a
subclass of "top", "person", and "organizationalPerson". You should add all of
these classes as values of the objectClass attribute. */
char *objectClass_values[] = { "top", "person", "organizationalPerson",
"inetOrgPerson", NULL };
...
/* Specify the value and type of each attribute in separate LDAPMod structures
*/
attribute1.mod_type = "sn";
attribute1.mod_values = sn_values;
attribute2.mod_type = "cn";
attribute2.mod_values = cn_values;
attribute3.mod_type = "objectClass";
attribute3.mod_values = objectClass_values;

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;
...
/* Set up the timeout period for adding the new entry */
tv.tv_sec = tv.tv_usec = 0;

/* Add the user "Barbara Jensen" */
if ( ( msgid = ldap_add( ld, dn, list_of_attrs ) ) == -1 ) {
  ldap_perror( ld, "ldap_add" );
  return( 1 );
}

/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
  ...
  /* do other work while waiting for the operation to complete */
  ...
}

/* Check the result to see if any errors occurred */
if (( rc = ldap_result2error( ld, result, 1 )) != LDAP_SUCCESS ) {
  printf( "Error while adding entry: %s\n", ldap_err2string( rc ));
}
...
```

See Also

`ldap_add_ext()`.

ldap_add_ext()

Adds a new entry to the directory asynchronously.

Syntax

```
#include <ldap.h>
int ldap_add_ext( LDAP *ld, const char *dn, LDAPMod **attrs,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-18 ldap_add_ext() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to add. With the exception of the leftmost component, all components of the distinguished name (for example, `o=`*organization* or `c=`*country*) must already exist.
`attrs`	Pointer to a `NULL`-terminated array of pointers to `LDAPMod` structures representing the attributes of the new entry.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`msgidp`	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call `ldap_result()` and `ldap_parse_result()` functions.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.

- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The ldap_add_ext() adds a new entry to the directory asynchronously.

This function is a new version of the ldap_add() function. If you are writing a new LDAP client, you should call this function instead of ldap_add().

To add a new entry to the directory, you need to specify the following information:

- A unique DN identifying the new entry.

  Use the dn argument to specify the DN of the new entry. Note that the parents of the entry should already exist. For example, if you are adding the entry uid=bjensen, ou=People, o=airius.com, the entries ou=People, o=airius.com and o=airius.com should already exist in the directory.

- A set of attributes for the new entry.

  Create an LDAPMod structure for each attribute. Set the mod_op field to 0 if the attribute values are string values. To specify values that consist of binary data (such as a sound file or a JPEG file), set the mod_op field to LDAP_MOD_BVALUES.

  Create an array of these LDAPMod structures and pass the array as the attrs argument.

ldap_add_ext() is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function ldap_add_ext_s() instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP add operation, you need to call the ldap_result() function and the ldap_parse_result() function. (See "Calling Asynchronous Functions" for details.) For a list of possible result codes for an LDAP add operation, see the result code documentation for the ldap_add_ext_s() function.

For additional information on adding new entries to the directory, see “Adding a New Entry.”

Example

See the example under “Example: Adding an Entry to the Directory (Asynchronous).”

See Also

`ldap_add_ext_s(), ldap_result(), ldap_parse_result(), LDAPMod.`

ldap_add_ext_s()

Adds a new entry to the directory synchronously.

Syntax

```
#include <ldap.h>
int ldap_add_ext_s( LDAP *ld, const char *dn, LDAPMod **attrs,
  LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-19 ldap_add_ext_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to add. With the exception of the leftmost component, all components of the distinguished name (for example, o=*organization* or c=*country*) must already exist.
`attrs`	Pointer to a NULL-terminated array of pointers to `LDAPMod` structures representing the attributes of the new entry.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

The following result codes can be returned by the Netscape Directory Server when processing an LDAP add request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- LDAP_OPERATIONS_ERROR may be sent by the Netscape Directory Server for general errors encountered by the server when processing the request.
- LDAP_PROTOCOL_ERROR if the add request sent by this function did not comply with the LDAP protocol (for example, if the server encountered an error when decoding your client's BER-encoded request).
- LDAP_CONSTRAINT_VIOLATION may be sent by the Netscape Directory Server if the server is configured to require a minimum password length and the new entry includes a value for the userpassword attribute that is shorter than the minimum length.

  The server may also send this result code if the value of the userpassword attribute is the same as the value of the uid, cn, sn, givenname, ou, or mail attributes. (Using a password that is the same as your user id or email address would make the password trivial and easy to crack.)
- LDAP_TYPE_OR_VALUE_EXISTS may be sent by the Netscape Directory Server if the set of attributes specified by the attrs argument includes duplicate attribute values.

- `LDAP_INVALID_DN_SYNTAX` may be sent by the Netscape Directory Server if the DN specified by the `dn` argument is not a valid DN.
- `LDAP_ALREADY_EXISTS` may be sent by the Netscape Directory Server if the DN specified by the `dn` argument identifies an entry already in the directory.
- `LDAP_OBJECT_CLASS_VIOLATION` may be sent by the Netscape Directory Server if the new entry does not comply with the Directory Server schema (for example, if one or more required attributes are not specified).
- `LDAP_NO_SUCH_OBJECT` may be sent by the Netscape Directory Server if the parent of the entry does not exist and if you are not authenticated as the root DN (for example, if you attempt to add `uid=bjensen, ou=People, o=airius.com` and if `ou=People, o=airius.com` does not exist).

  This result code may also be sent if the DN of the new entry has a suffix that is not handled by the current server and no referral URLs are available.
- `LDAP_REFERRAL` may be sent by the Netscape Directory Server if the DN specified by the `dn` argument identifies an entry not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is `uid=bjensen, ou=European Sales, o=airius.com`, all entries under `ou=European Sales` might be handled by a different Directory Server.)
- `LDAP_UNWILLING_TO_PERFORM` may be sent by the Netscape Directory Server if the server's database is set up to not allow write operations to the database (the database is read-only).
- `LDAP_INVALID_SYNTAX` may be sent by the Netscape Directory Server if the entry or the entry's parent has an invalid ACL.
- `LDAP_INSUFFICIENT_ACCESS` may be sent by the Netscape Directory Server in the following situations:
  - The ACL for the entry's parent does not allow you to add the entry.
  - The entry's parent has no ACL.
  - The entry has no parent and your client is not authenticated as the root DN.

Note that the Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The `ldap_add_ext_s()` function adds a new entry to the directory synchronously.

This function is a new version of the `ldap_add_s()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_add_s()`.

To add a new entry to the directory, you need to specify the following information:

- A unique DN identifying the new entry.

  Use the `dn` argument to specify the DN of the new entry. Note that the parents of the entry should already exist. For example, if you are adding the entry `uid=bjensen, ou=People, o=airius.com`, the entries `ou=People, o=airius.com` and `o=airius.com` should already exist in the directory.

- A set of attributes for the new entry.

  Create an `LDAPMod` structure for each attribute. Set the `mod_op` field to `0` if the attribute values are string values. To specify values that consist of binary data (such as a sound file or a JPEG file), set the `mod_op` field to `LDAP_MOD_BVALUES`.

  Create an array of these `LDAPMod` structures and pass the array as the `attrs` argument.

`ldap_add_ext_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_add_ext()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For additional information on adding new entries to the directory, see "Adding a New Entry."

Example

See the example under "Example: Adding an Entry to the Directory (Synchronous)."

See Also

`ldap_add_ext(), LDAPMod.`

ldap_add_s()

Adds a new entry to the directory synchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_add_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_add_s( LDAP *ld, const char *dn, LDAPMod **attrs );
```

Parameters

This function has the following parameters:

Table 18-20 ldap_add_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to add. With the exception of the leftmost component, all components of the distinguished name (for example, o=*organization* or c=*country*) must already exist.
`attrs`	Pointer to a `NULL`-terminated array of pointers to `LDAPMod` structures representing the attributes of the new entry.

Returns

See the result code documentation for the `ldap_add_ext_s()` function for a list of possible return codes for the LDAP add operation.

Description

The `ldap_add_s()` function adds a new entry to the directory synchronously.

A newer version of this function, `ldap_add_ext_s()`, is available in this release of the LDAP API. `ldap_add_s()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_add_ext_s()` instead of `ldap_add_s()`.

If you want more information on `ldap_add_s()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following example adds a new entry to the directory.

Code Example 18-5 ldap_add_s code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;

/* Distinguished name of the new entry. Note that "o=airius.com" and
"ou=People, o=airius.com" must already exist in the directory. */
char *dn = "uid=bjensen, ou=People, o=airius.com";

/* To add a "person" entry, you must specify values for the sn, cn, and
objectClass attributes. (These are required attributes.) */
char *sn_values[] = { "Jensen", NULL };

/* To specify multiple values for an attribute, add the different values to the
array. */
char *cn_values[] = { "Barbara Jensen", "Babs Jensen", NULL };

/* The object class for a "person" entry is "inetOrgPerson", which is a
subclass of "top", "person", and "organizationalPerson". You should add all of
these classes as values of the objectClass attribute. */
char *objectClass_values[] = { "top", "person", "organizationalPerson",
"inetOrgPerson", NULL };
...
/* Specify the value and type of each attribute in separate LDAPMod structures
*/
attribute1.mod_type = "sn";
attribute1.mod_values = sn_values;
attribute2.mod_type = "cn";
attribute2.mod_values = cn_values;
attribute3.mod_type = "objectClass";
attribute3.mod_values = objectClass_values;

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;
...
/* Add the user "Barbara Jensen" */
if ( ldap_add_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_add_s" );
  return( 1 );
}
...
```

See Also

`ldap_add_ext_s()`.

ldap_ber_free()

This function is documented here only for backward compatibility; you should use the `ber_free()` function in its place since this function will be phased out over time. Except in name, the function `ldap_ber_free()` is idendical to `ber_free()`.

Syntax

```
#include <ldap.h>
void ldap_ber_free( BerElement *ber, int freebuf );
```

Parameters

This function has the following parameters:

Table 18-21 ldap_ber_free() function parameters

`ber`	Pointer to the `BerElement` structure that you want to free.
`freebuf`	Specifies whether or not to free the buffer in the `BerElement` structure.

ldap_build_filter()

The `ldap_build_filter()` function is a deprecated function. Use the `ldap_create_filter()` function instead.

ldap_compare()

Asynchronously determines if an attribute of an entry contains a specified value.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_compare_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_compare( LDAP *ld, const char *dn, const char *attr,
  const char *value );
```

Parameters

This function has the following parameters:

Table 18-22 ldap_compare() function parameters

`ld`	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry used in the comparison.
`attr`	Attribute type that you want to check the value against.
`value`	Value that you want to compare against the attribute values.

Returns

Returns the message ID of the `ldap_compare()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`. For a list of possible return codes for the LDAP compare operation, see the result code documentation for the `ldap_compare_ext_s()` function.

Description

The `ldap_compare()` function compares a value with the value of an attribute in an entry.

A newer version of this function, `ldap_compare_ext()`, is available in this release of the LDAP API. `ldap_compare()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_compare_ext()` instead of `ldap_compare()`.

If you want more information on `ldap_compare()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code checks to see if Barbara Jensen has the e-mail address "`bjensen@airius.com`".

Code Example 18-6 Using ldap_compare()

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *dn = "uid=bjensen, ou=People, o=airius.com";
int msgid;
...
msg_id = ldap_compare( ld, dn, "mail", "bjensen@airius.com" );
...
```

See Also

ldap_compare_ext().

ldap_compare_ext()

Asynchronously determines if an attribute of an entry contains a specified value.

Syntax

```
#include <ldap.h>
int ldap_compare_ext( LDAP *ld, const char *dn,
  const char *attr, struct berval *bvalue,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-23 ldap_compare_ext() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	Distinguished name (DN) of the entry used in the comparison.
attr	Attribute type that you want to check the value against.
value	Value that you want to compare against the attribute values.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call the ldap_result() and ldap_parse_result() functions.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.

- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The ldap_compare_ext() function asynchronously compares the value of an attribute in an entry against a specified value.

This function is a new version of the ldap_compare() function. If you are writing a new LDAP client, you should call this function instead of ldap_compare().

ldap_compare_ext() is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function ldap_compare_ext_s() instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP compare operation, you need to call the ldap_result() function and the ldap_parse_result() function. (See "Calling Asynchronous Functions" for details.) For a list of possible result codes for an LDAP compare operation, see the result code documentation for the ldap_compare_ext_s() function.

For additional information on comparing attribute values in an entry, see "Comparing the Value of an Attribute."

Example

See the example under "Example: Comparing a Value in an Entry (Asynchronous)."

See Also

ldap_compare_ext_s(), ldap_result(), ldap_parse_result().

ldap_compare_ext_s()

Synchronously determines if an attribute of an entry contains a specified value.

Syntax

```
#include <ldap.h>
int ldap_compare_ext_s( LDAP *ld, const char *dn,
  const char *attr, struct berval *bvalue,
  LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-24 ldap_compare_ext_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry used in the comparison.
`attr`	Attribute type that you want to check the value against.
`value`	Value that you want to compare against the attribute values.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Returns

One of the following values:

- `LDAP_COMPARE_TRUE` if the entry contains the attribute value.
- `LDAP_COMPARE_FALSE` if the entry does not contain the attribute value.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.

- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

The following result codes can be returned by the Netscape Directory Server when processing an LDAP compare request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- LDAP_OPERATIONS_ERROR may be sent by the Netscape Directory Server for general errors encountered by the server when processing the request.
- LDAP_PROTOCOL_ERROR if the compare request sent by this function did not comply with the LDAP protocol (for example, if the server encountered an error when decoding your client's BER-encoded request).
- LDAP_NO_SUCH_OBJECT may be sent by the Netscape Directory Server if the specified entry has a suffix that is not handled by the current server and no referral URLs are available.
- LDAP_REFERRAL may be sent by the Netscape Directory Server if the DN specified by the dn argument identifies an entry not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is uid=bjensen, ou=European Sales, o=airius.com, all entries under ou=European Sales might be handled by a different Directory Server.)
- LDAP_INSUFFICIENT_ACCESS may be sent by the Netscape Directory Server if your client does not have the access right to compare this entry.
- LDAP_INVALID_SYNTAX may be sent by the Netscape Directory Server if the entry or the entry's parent has an invalid ACL.
- LDAP_NO_SUCH_ATTRIBUTE may be sent by the Netscape Directory Server if the entry does not contain the attribute specified by the attr argument.

Note that the Netscape Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The ldap_compare_ext_s() function synchronously compares the value of an attribute in an entry against a specified value.

This function is a new version of the `ldap_compare_s()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_compare_s()`.

`ldap_compare_ext_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_compare_ext()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For additional information on comparing attribute values in an entry, see "Comparing the Value of an Attribute."

Example

See the example under "Example: Comparing a Value in an Entry (Synchronous)."

See Also

`ldap_compare_ext()`.

ldap_compare_s()

Synchronously determines if an attribute of an entry contains a specified value.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_compare_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_compare_s( LDAP *ld, const char *dn,
  const char *attr, const char *value );
```

Parameters

This function has the following parameters:

Table 18-25 ldap_compare_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry used in the comparison.
`attr`	Attribute type that you want to check the value against.
`value`	Value that you want to compare against the attribute values.

Returns

For a list of the possible result codes for an LDAP compare operation, see the result code documentation for the `ldap_compare_ext_s()` function.

Description

The `ldap_compare_s()` function compares a value with the value of an attribute in an entry.

A newer version of this function, `ldap_compare_ext_s()`, is available in this release of the LDAP API. `ldap_compare_s()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_compare_ext_s()` instead of `ldap_compare_s()`.

If you want more information on `ldap_compare_s()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code checks to see if Barbara Jensen has the e-mail address "`bjensen@airius.com`".

Code Example 18-7 ldap_compare_s() code example

```
#include <stdio.h>
#include <ldap.h>
LDAP *ld;
char *dn = "uid=bjensen, ou=People, o=airius.com";
int has_value;
...
has_value = ldap_compare_s( ld, dn, "mail", "bjensen@airius.com" );
switch ( has_value ) {
  case LDAP_COMPARE_TRUE:
    printf( "The mail attribute contains bjensen@airius.com.\n");
    break;
  case LDAP_COMPARE_FALSE:
    printf( "The mail attribute does not contain bjensen@airius.com.\n");
    break;
  default:
    ldap_perror( ld, "ldap_compare_s" );
    return( 1 );
}
...
```

See Also

`ldap_compare_ext_s()`.

ldap_control_free()

Frees an LDAPControl structure from memory.

Syntax

```
#include <ldap.h>
void ldap_control_free( LDAPControl *ctrl );
```

Parameters

This function has the following parameters:

Table 18-26 ldap_control_free() function parameters

ctrl	Pointer to an LDAPControl structure that you want to free from memory.

Description

The ldap_control_free() function frees an LDAPControl structure from memory.

You should call this function to free controls that you create (for example, if you call the ldap_create_sort_control() function).

See Also

ldap_controls_free().

ldap_controls_free()

Frees an array of LDAPControl structures from memory.

Syntax

```
#include <ldap.h>
void ldap_controls_free( LDAPControl **ctrls );
```

Parameters

This function has the following parameters:

Table 18-27 ldap_controls_free() function parameters

ctrls	Pointer to an array of LDAPControl structures that you want to free from memory.

Description

The `ldap_controls_free()` function frees an array of `LDAPControl` structures from memory.

You should call this function to free arrays of controls that you create or any arrays returned by `ldap_parse_result()`.

See Also

`ldap_control_free()`.

ldap_count_entries()

Returns the number of `LDAPMessage` structures representing directory entries in a chain of search results.

Syntax

```
#include <ldap.h>
int ldap_count_entries( LDAP *ld, LDAPMessage *result );
```

Parameters

This function has the following parameters:

Table 18-28 ldap_count_entries() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`result`	Chain of search results, represented by the pointer to an `LDAPMessage` structure.

Returns

One of the following values:

- The number of `LDAPMessage` structures of the type `LDAP_RES_SEARCH_ENTRY` in a chain of search results, if successful. (If there are no structures of this type, returns `0`.)
- -1 if `ld` is not a valid connection handle.

Description

The `ldap_count_entries()` function returns the number of `LDAPMessage` structures representing directory entries in a chain of search results. These messages have the type `LDAP_RES_SEARCH_ENTRY`.

Note that if you pass in a pointer to an LDAPMessage structure in the middle of the chain of results, the function counts only the entries between that structure and the last structure in the chain. In this type of situation, the function does not return the count of all entries in the chain.

For more information on using this function, see "Iterating Through a Chain of Results."

Example

See the examples under ldap_search_ext() and ldap_search_ext_s().

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(), ldap_first_entry(), ldap_next_entry(), ldap_first_message(), ldap_next_message().

ldap_count_messages()

Returns the number of LDAPMessage structures in a chain of search results.

Syntax

```
#include <ldap.h>
int ldap_count_messages( LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the following parameters:

Table 18-29 ldap_count_messages() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
result	Chain of search results, represented by the pointer to an LDAPMessage structure.

Returns

One of the following values:

- The number of LDAPMessage structures in a chain of search results, if successful. (If there are no structures, returns 0.)
- -1 if ld is not a valid connection handle.

Description

The ldap_count_messages() function returns the number of LDAPMessage structures in a chain of search results.

Note that if you pass in a pointer to an LDAPMessage structure in the middle of the chain of results, the function counts only between that structure and the last structure in the chain. In this type of situation, the function does not return the count of all structures in the chain.

For more information on using this function, see "Iterating Through a Chain of Results."

Example

See the examples under ldap_search_ext() and ldap_search_ext_s().

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(), ldap_first_message(), ldap_next_message().

ldap_count_references()

Returns the number of LDAPMessage structures representing search references in a chain of search results.

Syntax

```
#include <ldap.h>
int ldap_count_references( LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the following parameters:

Table 18-30 ldap_count_references() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
result	Chain of search results, represented by the pointer to an LDAPMessage structure.

Returns

One of the following values:

- The number of LDAPMessage structures of the type LDAP_RES_SEARCH_REFERENCE in a chain of search results, if successful. (If there are no structures of this type, returns 0.)
- -1 if ld is not a valid connection handle.

Description

The ldap_count_references() function returns the number of LDAPMessage structures representing search references in a chain of search results. These messages have the type LDAP_RES_SEARCH_REFERENCE.

Note that if you pass in a pointer to an LDAPMessage structure in the middle of the chain of results, the function counts only the references between that structure and the last structure in the chain. In this type of situation, the function does not return the count of all references in the chain.

For more information on using this function, see "Iterating Through a Chain of Results."

Example

See the examples under ldap_search_ext() and ldap_search_ext_s().

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(), ldap_first_reference(), ldap_next_reference().

ldap_count_values()

The ldap_count_values() function returns the number of values in an array of strings. Use the ldap_count_values_len() function instead of this function if the array contains berval structures.

For additional information, see "Getting the Values of an Attribute."

Syntax

```
#include <ldap.h>
int ldap_count_values( char **values );
```

Parameters

This function has the following parameters:

Table 18-31 ldap_count_values() function parameters

values	Array of values.

Returns

One of the following values:

- The number of values in the array, if successful.
- `-1` if unsuccessful. (See Chapter 19, "Result Codes" for a complete listing.)

Example

The following section of code counts the number of values assigned to an attribute.

Code Example 18-8 ldap_count_values() code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *e;
char *a="cn";
char **vals;
int count;
...

/* Get the values of the cn attribute */
vals = ldap_get_values( ld, e, a );

/* Count the values of the attribute */
count = ldap_count_values( vals );
...
```

See Also

`ldap_count_values_len(), ldap_get_values().`

ldap_count_values_len()

The `ldap_count_values_len()` function returns the number of values in an array of `berval` structures. Use the `ldap_count_values()` function instead of this function if the array contains strings.

For additional information, see "Getting the Values of an Attribute."

Syntax

```
#include <ldap.h>
int ldap_count_values_len( struct berval **vals );
```

Parameters

This function has the following parameters:

Table 18-32 ldap_count_values_len() function parameters

`values`	Array of berval structures.

Returns

One of the following values:

- The number of values in the array, if successful.
- `-1` if unsuccessful. (See Chapter 19, “Result Codes” for a complete listing.)

Example

The following section of code counts the number of values assigned to an attribute.

Code Example 18-9 ldap_count_values_len() code example

```
#include <ldap.h>
LDAP *ld;
LDAPMessage *e;
char *a="jpegPhoto";
struct berval **bvals;
int count;
...
/* Get the values of the jpegPhoto attribute */
bvals = ldap_get_values_len( ld, e, a );

/* Count the values of the attribute */
count = ldap_count_values_len( vals );
...
```

See Also

`ldap_count_values(), ldap_get_values_len().`

ldap_create_filter()

The `ldap_create_filter()` routine constructs an LDAP search filter. For more information about filters, see “Creating Filters Programmatically.”

Syntax

```
#include <ldap.h>
int ldap_create_filter( char *buf, unsigned long buflen,
  char *pattern, char *prefix, char *suffix, char *attr,
  char *value, char **valwords );
```

Parameters

This function has the following parameters:

Table 18-33 ldap_create_filter() function parameters

`buf`	Buffer to contain the constructed filter.
`buflen`	Size of the buffer.
`pattern`	Pattern for the filter.
`prefix`	Prefix to prepend to the filter (`NULL` if not used).
`suffix`	Suffix to append to the filter (`NULL` if not used).
`attr`	Replaces `%a` in the pattern.
`value`	Replaces `%v` in the pattern.
`valwords`	Replaces `%v`*M* through `%v`*N* in the pattern.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_SIZELIMIT_EXCEEDED` if the created filter exceeds the size of the buffer.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.

Example

The following section of code builds the filter `(mail=bjensen@airius.com)`.

Code Example 18-10 Creating a filter with ldap_create_filter()

```
char buf[LDAP_FILT_MAXSIZ];
char *pattern = "(%a=%v)";
char *attr = "mail";
char *value = "bjensen@airius.com";
...
ldap_create_filter( buf, LDAP_FILT_MAXSIZ, pattern, NULL,
  NULL, attr, value, NULL );
...
```

See Also

ldap_init_getfilter(), ldap_init_getfilter_buf(), ldap_getfirstfilter(), ldap_getnextfilter(), ldap_set_filter_additions().

ldap_create_persistentsearch_control()

Creates a control that allows your client to perform a persistent search of an LDAP v3 server, which allows the search operation to continue without termination until your client abandons the search.

This function implements an extension to the LDAPv3 protocol. Persistent search is an optional LDAP server feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension.

Syntax

```
#include <ldap.h>
int ldap_create_persistentsearch_control( LDAP *ld,
  int changetypes, int changesonly, int return_echg_ctls,
  char ctl_iscritical, LDAPControl **ctrlp );
```

Parameters

This function has the following parameters:

Table 18-34 ldap_create_persistentsearch_control() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 18-34 ldap_create_persistentsearch_control() function parameters

`changetypes`	Specifies the types of changes that you want to keep track of. This field can have one or more of the following values (you can OR the values together to specify multiple types): • `LDAP_CHANGETYPE_ADD` specifies that you want to keep track of entries added to the directory. • `LDAP_CHANGETYPE_DELETE` specifies that you want to keep track of entries deleted from the directory. • `LDAP_CHANGETYPE_MODIFY` specifies that you want to keep track of entries that are modified. • `LDAP_CHANGETYPE_MODDN` specifies that you want to keep track of entries that are renamed. • `LDAP_CHANGETYPE_ANY` specifies that you want to keep track of all of the above changes to the directory.
`changesonly`	Specifies whether or not you want skip the initial search and only get the latest changes as they occur: • If non-zero, the initial search is skipped and only entries that have changed after the initial search are returned. • If `0`, the results of the initial search are returned first.
`return_echg_ctls`	Specifies whether or not entry change notification controls are included with each entry returned to your client: • If non-zero, an entry change notification control is included with each entry. • If `0`, entry change notification controls are not included with the entries returned from the server.
`ctl_iscritical`	Specifies whether or not the persistent search control is critical to the search operation: • If non-zero, the control is critical to the search operation. If the server does not support persistent searches, the server will return the error `LDAP_UNAVAILABLE_CRITICAL_EXTENSION`. • If `0`, the control is not critical to the search operation. Even if the server does not support persistent searches, the search operation is still performed.
`ctrlp`	Pointer to a pointer to an `LDAPControl` structure that will be created by this function. When you are done using this control, you should free it by calling the `ldap_control_free()` function.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the control.

Description

The `ldap_create_persistentsearch_control()` function allows you to perform persistent searches. A persistent search provides the means to track changes to a set of entries that match the search criteria. After the initial search is performed, the server keeps track of the search criteria and sends back information when any entry that matches the criteria is added, deleted, modified, or renamed.

Calling this function creates an LDAP server control that you can pass to the `ldap_search_ext()` function.

In order for the control to work, the LDAP server that you are connecting to must support the server control for persistent searches (OID 2.16.840.1.113730.3.4.3, or `LDAP_CONTROL_PERSISTENTSEARCH`, as defined in the `ldap.h` header file). See "Determining the Controls Supported By the Server" for information on determining the controls supported by a server.

After you create the control, you can pass it to the LDAP server during a search operation. (Pass the server control when calling the `ldap_search_ext()` function.) If you specify that you want "entry change notification" controls sent back (that is, if you specify a non-zero value for the `return_echg_ctls` argument), the server includes controls with each changed entry it sends back.

To retrieve the "entry change notification control" from each entry, call the `ldap_get_entry_controls()` function. To get data about the changes made to the entry from the control, call the `ldap_parse_entrychange_control()` function.

When you are done with the search, you can cancel the persistent search by calling the `ldap_abandon_ext()` function. You should also free the control from memory by calling the `ldap_control_free()` function.

See Also

`ldap_search_ext()`, `ldap_abandon_ext()`, `ldap_get_entry_controls()`, `ldap_parse_entrychange_control()`, `ldap_control_free()`.

ldap_create_proxyauth_control()

You use `ldap_create_proxyauth_control()` to create an LDAPv3 control that allows a bound entity to assume the identity of another directory entry.

This function implements the proxy authorization extension of the LDAPv3 protocol. Proxy authorization is an optional LDAP server feature and it may not be supported on all LDAP servers.

Syntax

```
#include <ldap.h>
int ldap_create_proxyauth_control( LDAP *ld, char *DN,
  char ctl_iscritical, LDAPControl **ctrlp);
```

Parameters

This function has the following parameters:

Table 18-35

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`DN`	String representing the distinguished name of the entry who's identity the client will be assuming.
`ctl_iscritical`	Specifies whether the persistent search control is critical to the search operation. For proxy authorization controls, this should be set to a non-zero value. If non-zero, the control is critical to the directory operation. If the server does not support proxy authentication, the server will return an `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` error. If `0`, the control is not critical to the directory operation. Even if the server does not support proxied authorization, the operation is still attempted and the proxied authorization control is ignored.
`ctrlp`	Pointer to a pointer to an `LDAPControl` structure that is created by this function. When you are done using this control, you should free it by calling the `ldap_control_free()` function.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.

- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_UNAVAILABLE_CRITICAL_EXTENSION if the server does not support proxied authorization and ctl_iscritical is set to a non-zero value.

See Also

ldap_control_free()

ldap_create_sort_control()

Creates a control that specifies the order in which you want search results returned.

This function implements an extension to the LDAPv3 protocol. Server-side sorting is an optional LDAP server feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension.

Syntax

```
#include <ldap.h>
int ldap_create_sort_control( LDAP *ld,
  LDAPsortkey **sortKeyList, const char ctl_iscritical,
  LDAPControl **ctrlp );
```

Parameters

This function has the following parameters:

Table 18-36 ldap_create_sort_control() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
sortKeyList	Pointer to an array of LDAPsortkey structures that specify the attribute types or matching rules used for sorting and the order (ascending or descending) in which to sort the results.
ctl_iscritical	Specifies whether or not the control is critical to the search operation. This field can have one of the following values: • A nonzero value specifies that the control is critical to the operation. • 0 specifies that the control is not critical to the operation.
ctrlp	Pointer to a pointer to an LDAPControl structure that will be created by this function. When you are done using this control, you should free it by calling the ldap_control_free() function.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.

Description

The ldap_create_sort_control() function allows you to specify the order in which you want to receive data from the server. Calling this function creates an LDAP control that you can pass to the ldap_search_ext() and ldap_search_ext_s() functions.

In order for the control to work, the LDAP server that you are connecting to must support the server control for sorting search results (OID 1.2.840.113556.1.4.473, or LDAP_CONTROL_SORTREQUEST as defined in ldap.h). See "Determining the Controls Supported By the Server" for information on determining the controls supported by a server.

To specify the attributes to use for sorting the results, you can call ldap_create_sort_keylist() to create an array of LDAPsortkey structures and pass the array as the sortKeyList argument.

When you are done with the search, you should free the control and the array of LDAPsortkey structures by calling the ldap_control_free() function and the ldap_free_sort_keylist() function.

See Also

ldap_create_sort_keylist(), ldap_search_ext(), ldap_search_ext_s(), ldap_control_free().

ldap_create_sort_keylist()

Creates an array of LDAPsortkey structures from a string representation of a set of sort keys.

Syntax

```
#include <ldap.h>
int ldap_create_sort_keylist(LDAPsortkey ***sortKeyList,
  const char *string_rep);
```

Parameters

This function has the following parameters:

Table 18-37 ldap_create_sort_keylist() function parameters

`sortKeyList`	Pointer to an array of `LDAPsortkey` structures that specify the attribute types or matching rules used for sorting and the order (ascending or descending) in which to sort the results.
`string_rep`	String representation of a set of sort keys.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `-1` if an error occurred

Description

The `ldap_create_sort_keylist()` function allows you to create an array of `LDAPsortkey` structures from a string representation of a set of sort keys. Calling this function creates an array of `LDAPsortkey` structures that you can pass to the `ldap_create_sort_control()` function.

The string representation specified by the `string_rep` argument should specify the name of the attribute that you want to sort by.

- To sort in reverse order, precede the attribute name with a hyphen ("-").
- To use a matching rule for sorting, append a colon to the attribute name and specify the object identifier (OID) of a matching rule after the colon.

For example:

- `cn` (sort by the cn attribute)
- `-cn` (sort by the cn attribute in reverse order)
- `-cn:1.2.3.4` (sort by the cn attribute in reverse order and use the matching rule identified by the OID 1.2.3.4)

When you are done sorting the results, you should free the array of `LDAPsortkey` structures by calling the `ldap_free_sort_keylist()` function.

See Also

`ldap_create_sort_control(), ldap_free_sort_keylist().`

ldap_create_virtuallist_control()

Creates a control that requests a subset of search results for use in a virtual list box.

This function implements an extension to the LDAPv3 protocol. This control is supported by the Netscape Directory Server, version 4.0 and later; and all iPlanet Directory Server versions. For information on determining if a server supports this or other LDAPv3 controls, see "Determining If the Server Supports LDAPv3," on page 211.

Syntax

```
#include <ldap.h>
int ldap_create_virtuallist_control( LDAP *ld,
  LDAPVirtualList *ldvlistp, LDAPControl **ctrlp );
```

Parameters

This function has the following parameters:

Table 18-38 ldap_create_virtuallist_control() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`ldvlistp`	Pointer to an `LDAPVirtualList` structure that specifies the subset of entries that you want retrieved from the server and the selected entry.
`ctrlp`	Pointer to a pointer to an `LDAPControl` structure that will be created by this function. When you are done using this control, you should free it by calling the `ldap_control_free()` function.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the control

Description

The `ldap_create_virtuallist_control()` function allows you to retrieve a subset of entries from the server for use in a virtual list box. Calling this function creates an LDAP control that you can pass to the `ldap_search_ext()` and `ldap_search_ext_s()` functions.

Note that you also need to pass a server-side sorting control to the search functions. You can call the `ldap_create_sort_keylist()` and `ldap_create_sort_control()` functions to create a server-side sorting control.

In order for the virtual list view control to work, the LDAP server that you are connecting to must support the server control for sorting search results (OID 2.16.840.1.113730.3.4.9, or `LDAP_CONTROL_VLVREQUEST`, as defined in `ldap.h`).

At this point in time, no server supports this control. (The Netscape Directory Server 3.x does not support this control, although future releases of the Directory Server plan to support it.) For information on determining if a server supports this or other LDAPv3 controls, see “Determining If the Server Supports LDAPv3,” on page 211.

To specify the subset of entries that you want to retrieve, create an `LDAPVirtualList` structure and pass in a pointer to this structure as the `ldvlistp` argument.

When you are done with the search, you should free the control by calling the `ldap_control_free()` function.

For more information about this control, see “Using the Virtual List View Control.”

See Also

`LDAPVirtualList, ldap_parse_virtuallist_control(), ldap_search_ext(), ldap_search_ext_s(), ldap_control_free().`

ldap_delete()

Deletes an entry from the directory asynchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_delete_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_delete( LDAP *ld, const char *dn );
```

Parameters

This function has the following parameters:

Table 18-39 ldap_delete() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to remove.

Returns

Returns the message ID of the `ldap_delete()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`. For a list of the possible result codes for an LDAP delete operation, see the result code documentation for the `ldap_delete_ext_s()` function.

Description

The `ldap_delete()` function removes an entry from the directory.

A newer version of this function, `ldap_delete_ext()`, is available in this release of the LDAP API. `ldap_delete()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_delete_ext()` instead of `ldap_delete()`.

If you want more information on `ldap_delete()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code uses the asynchronous `ldap_delete()` function to remove the entry for "`Barbara Jensen`" from the directory.

Code Example 18-11 ldap_delete() code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the entry that you want to delete. */
char *dn = "uid=bjensen, ou=People, o=airius.com";
...
/* Set up the timeout period to wait for the "modify" operation */
tv.tv_sec = tv.tv_usec = 0;

/* Delete the entry */
```

Code Example 18-11 ldap_delete() code example

```
if ( ( msgid = ldap_delete( ld, dn ) ) == -1 ) {
  ldap_perror( ld, "ldap_delete" );
  return( 1 );
}
/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
  ...
  /* do other work while waiting for the operation to complete */
  ...
}
/* Check the result to see if any errors occurred */
ldap_result2error( ld, result, 1 );
ldap_perror( ld, "ldap_delete" );
...
```

See Also

`ldap_delete_ext()`.

ldap_delete_ext()

Deletes an entry from the directory asynchronously.

Syntax

```
#include <ldap.h>
int ldap_delete_ext( LDAP *ld, const char *dn,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-40 ldap_delete_ext() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to remove.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Table 18-40 ldap_delete_ext() function parameters

`msgidp`	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call the `ldap_result()` and `ldap_parse_result()` functions.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The `ldap_delete_ext()` function deletes an entry from the directory asynchronously. Use the `dn` argument to specify the entry that you want to delete.

This function is a new version of the `ldap_delete()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_delete()`.

`ldap_delete_ext()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_delete_ext_s()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP delete operation, you need to call the `ldap_result()` function and the `ldap_parse_result()` function. (See "Calling Asynchronous Functions" for details.) For a list of possible result codes for an LDAP delete operation, see the result code documentation for the `ldap_delete_ext_s()` function.

For additional information on deleting entries from the directory, see “Deleting an Entry.”

Example

See the example under “Example: Deleting an Entry from the Directory (Asynchronous).”

See Also

`ldap_delete_ext_s(), ldap_result(), ldap_parse_result().`

ldap_delete_ext_s()

Deletes an entry from the directory synchronously.

Syntax

```
#include <ldap.h>
int ldap_delete_ext_s( LDAP *ld, const char *dn,
  LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-41 ldap_delete_ext_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to remove.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.

- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

The following result codes can be returned by the Directory Server when processing an LDAP delete request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- LDAP_OPERATIONS_ERROR may be sent by the Directory Server for general errors encountered by the server when processing the request.
- LDAP_PROTOCOL_ERROR if the delete request did not comply with the LDAP protocol. The Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
  - The server encountered an error when decoding your client's BER-encoded request.
- LDAP_UNWILLING_TO_PERFORM may be sent by the Directory Server in the following situations:
  - The entry to be deleted is a DSE (DSA-specific entry, where DSA is the Directory Server Agent).
  - The server's database is read-only.
- LDAP_NO_SUCH_OBJECT may be sent by the Directory Server if the entry that you want deleted does not exist and if no referral URLs are available.
- LDAP_REFERRAL may be sent by the Directory Server if the DN specified by the dn argument identifies an entry not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is uid=bjensen, ou=European Sales, o=airius.com, all entries under ou=European Sales might be handled by a different Directory Server.)

- `LDAP_NOT_ALLOWED_ON_NONLEAF` may be sent by the Directory Server if the entry that you want deleted has entries beneath it in the directory tree (in other words, if this entry is a parent entry to other entries).
- `LDAP_INSUFFICIENT_ACCESS` may be sent by the Directory Server if the DN that your client is authenticated as does not have the access rights to write to the entry.

Note that the Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The `ldap_delete_ext_s()` function deletes an entry from the directory synchronously. Use the `dn` argument to specify the entry that you want to delete.

This function is a new version of the `ldap_delete_s()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_delete_s()`.

`ldap_delete_ext_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_delete_ext()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For additional information on deleting entries from the directory, see "Deleting an Entry."

Example

See the example under "Example: Deleting an Entry from the Directory (Synchronous)."

See Also

`ldap_delete_ext()`.

ldap_delete_s()

Deletes an entry from the directory synchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_delete_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_delete_s(LDAP *ld, const char *dn);
```

Parameters

This function has the following parameters:

Table 18-42 ldap_delete_s() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	Distinguished name (DN) of the entry to remove.

Returns

For a list of possible result codes for an LDAP delete operation, see the result code documentation for the ldap_delete_ext_s() function.

Description

The ldap_delete_s() function removes an entry from the directory.

A newer version of this function, ldap_delete_ext_s(), is available in this release of the LDAP API. ldap_delete_s() (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use ldap_delete_ext_s() instead of ldap_delete_s().

If you want more information on ldap_delete_s(), refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code uses the synchronous ldap_delete_s() function to delete the entry for Barbara Jensen from the directory.

Code Example 18-12 ldap_delete_s() code example

```
#include <ldap.h>
LDAP *ld;

/* Distinguished name of the entry that you want to delete. */
char *dn = "uid=bjensen, ou=People, o=airius.com";
...
/* Delete the entry */
if ( ldap_delete_s( ld, dn ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_delete_s" );
```

Code Example 18-12 ldap_delete_s() code example

```
#include <ldap.h>
  return( 1 );
}
...
```

See Also

`ldap_delete_ext_s()`.

ldap_dn2ufn()

The `ldap_dn2ufn()` function converts a distinguished name (DN) into a "friendlier" form by stripping off the cryptic type names.

Syntax

```
#include <ldap.h>
char * ldap_dn2ufn( const char *dn );
```

Parameters

This function has the following parameters:

Table 18-43 ldap_dn2ufn() function parameters

dn	Distinguished name (DN) that you want converted to a friendlier form.

Returns

One of the following values:

- If successful, returns the DN in its friendlier form.
- If unsuccessful, returns `NULL`.

ldap_err2string()

The `ldap_err2string()` function returns the corresponding error message for an error code. For more information, see “Getting the Error Message.”

Syntax

```
#include <ldap.h>
char * ldap_err2string( int err );
```

Parameters

This function has the following parameters:

Table 18-44 ldap_err2string() function parameters

`err`	Error code that you want interpreted into an error message.

Returns

One of the following values:

- If successful, returns the corresponding error message for the error code.
- If unsuccessful (for example, if the error code is not a valid LDAP API error code), returns "`Unknown error`".

Example

The following section of code sets the variable `err_msg` to the error message corresponding to the error code returned by the `ldap_simple_bind_s()` function.

Code Example 18-13 ldap_err2string() code example

```
#include <ldap.h>
...
LDAP *ld;
char *dn = "uid=bjensen, ou=People, o=airius.com";
char *pw = "hifalutin";
char *err_msg;
...
err_msg = ldap_err2string( ldap_simple_bind_s( ld, dn, pw ) );
...
```

See Also

`ldap_get_lderrno()`, `ldap_perror()`, `ldap_result2error()`, `ldap_set_lderrno()`, `ldapssl_err2string()`.

ldap_explode_dn()

The `ldap_explode_dn()` function converts a distinguished name (DN) into its component parts. For more information, see "Getting the Components of a Distinguished Name."

Syntax

```
#include <ldap.h>
char ** ldap_explode_dn( const char *dn, int notypes );
```

Parameters

This function has the following parameters:

Table 18-45 ldap_explode_dn() function parameters

`dn`	Distinguished name (DN) that you want separated into components.
`notypes`	Specifies whether or not type names in the distinguished name are returned. This parameter can have the following possible values: • `0` specifies that type names are returned. • A non-zero value specifies that type names are not returned.

Returns

One of the following values:

- If successful, returns a NULL-terminated array containing the components of the distinguished name (DN).
- If unsuccessful, returns a `NULL`.

Example

The following function call:

```
ldap_explode_dn( "uid=bjensen, ou=People, o=airius.com", 0 )
```

returns this array:

```
{ "uid=bjensen", "ou=People", "ou=airius.com", NULL }
```

If you change the `notypes` parameter from `0` to `1`:

```
ldap_explode_dn( "uid=bjensen, ou=People, o=airius.com", 1 )
```

the component names are not returned in the array:

```
{ "bjensen", "People", "airius.com", NULL }
```

See Also

ldap_explode_rdn(), ldap_get_dn().

ldap_explode_rdn()

The ldap_explode_rdn() function converts a relative distinguished name (RDN) into its component parts. For more information, see "Getting the Components of a Distinguished Name."

Syntax

```
#include <ldap.h>
char ** ldap_explode_rdn( const char *dn, int notypes );
```

Parameters

This function has the following parameters:

Table 18-46 ldap_explode_rdn() function parameters

dn	Relative distinguished name (RDN) that you want separated into components.
notypes	Specifies whether or not type names in the relative distinguished name are returned. This parameter can have the following possible values: • 0 specifies that type names are returned. • A non-zero value specifies that type names are not returned.

Returns

One of the following values:

- If successful, returns a NULL-terminated array containing the components of the relative distinguished name (RDN).
- If unsuccessful, returns a NULL.

Example

The following function call:

```
ldap_explode_rdn( "ou=Sales + cn=Barbara Jensen", 0 );
```

returns this array:

```
{ "ou=Sales", "cn=Barbara Jensen", NULL }
```

See Also

ldap_explode_dn(), ldap_get_dn().

ldap_extended_operation()

Sends a request to the server to perform an extended operation asynchronously.

Syntax

```
#include <ldap.h>
int ldap_extended_operation( LDAP *ld, const char *requestoid,
  struct berval *requestdata, LDAPControl **serverctrls,
  LDAPControl **clientctrls, int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-47 ldap_extended_operation() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
requestoid	Object identifier (OID) of the extended operation that you want the server to perform.
requestdata	Pointer to a berval structure containing the data that you want passed to the server to perform the extended operation.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call the ldap_result() and ldap_parse_result() functions.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.

- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_NOT_SUPPORTED` if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The `ldap_extended_operation()` function sends a request to the server to perform an LDAPv3 extended operation synchronously.

The LDAP server must support the extended operation. The Netscape Directory Server 3.0 supports a server plug-in interface that you can use to add support for extended operations to the server. For details, see the *Netscape Directory Server 3.0 Programmer's Guide*.

For information on determining the extended operations supported by a server, see "Determining the Extended Operations Supported."

After processing an LDAPv3 extended operation, an LDAP server can return an object identifier and data in the result. To parse the OID and data from the result, call the `ldap_parse_extended_result()` function.

`ldap_extended_operation()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_extended_operation_s()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP extended operation, you need to call the `ldap_result()` function, the `ldap_parse_extended_result()` function, and the `ldap_get_lderrno()` function. (See "Performing an Asynchronous Extended Operation" for details.) For a list of possible result codes for an LDAP extended operation, see the result code documentation for the `ldap_extended_operation_s()` function.

See Also

`ldap_extended_operation_s()`, `ldap_result()`, `ldap_parse_extended_result()`, `ldap_get_lderrno()`.

ldap_extended_operation_s()

Sends a request to the server to perform an extended operation synchronously.

Syntax

```
#include <ldap.h>
int ldap_extended_operation_s( LDAP *ld, const char *requestoid,
  struct berval *requestdata, LDAPControl **serverctrls,
  LDAPControl **clientctrls, char **retoidp,
  struct berval **retdatap );
```

Parameters

This function has the following parameters:

Table 18-48 ldap_extended_operation_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`requestoid`	Object identifier (OID) of the extended operation that you want the server to perform.
`requestdata`	Pointer to a `berval` structure containing the data that you want passed to the server to perform the extended operation.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`retoidp`	Pointer to the object identifier (OID) returned by the server after performing the extended operation. When done, you can free this by calling the `ldap_memfree()` function.
`retdatap`	Pointer to the pointer to a `berval` structure containing the data returned by the server after performing the extended operation. When done, you can free this by calling the `ber_bvfree()` function.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.

- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.
- `LDAP_NOT_SUPPORTED` if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

The following result codes can be returned by the Netscape Directory Server when processing an LDAP extended operation request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- `LDAP_OPERATIONS_ERROR` may be sent by the Netscape Directory Server for general errors encountered by the server when processing the request.
- `LDAP_PROTOCOL_ERROR` if the extended operation request did not comply with the LDAP protocol. The Netscape Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
  - The server encountered an error when decoding your client's BER-encoded request.
  - The extended operation specified by the `requestoid` argument is not supported by the server.

Depending on the extended operation requested, the Netscape Directory Server may send other result codes in addition to the codes described here. In the Netscape Directory Server, the people deploying the server are responsible for implementing the mechanisms for handling extended operations. Check with your server administrator for additional result codes returned to the client.

Description

The `ldap_extended_operation_s()` function sends a request to the server to perform an LDAPv3 extended operation synchronously.

The LDAP server must support the extended operation. The Netscape Directory Server 3.0 supports a server plug-in interface that you can use to add support for extended operations to the server. For details, see the *Netscape Directory Server 3.0 Programmer's Guide*.

For information on determining the extended operations supported by a server, see "Determining the Extended Operations Supported."

After processing an LDAPv3 extended operation, an LDAP server can return an object identifier and data in the results. The `retoidp` and `retdatap` arguments point to these values.

`ldap_extended_operation_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_extended_operation()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

See Also

`ldap_extended_operation()`.

ldap_first_attribute()

The `ldap_first_attribute()` function returns the name of the first attribute in a entry returned by the `ldap_first_entry()` function, the `ldap_next_entry()` function, or the `ldap_result()` function.

For more information, see "Getting Attributes from an Entry."

Syntax

```
#include <ldap.h>
char * ldap_first_attribute( LDAP *ld, LDAPMessage *entry,
  BerElement **ber );
```

Parameters

This function has the following parameters:

Table 18-49 ldap_first_attribute() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`entry`	Pointer to the LDAPMessage structure representing the entry returned by the `ldap_first_entry()` or `ldap_next_entry()` function.

Table 18-49 ldap_first_attribute() function parameters

ber	A pointer to a `BerElement` allocated to keep track of its current position. Pass this pointer to subsequent calls to `ldap_next_attribute()` to step through the entry's attributes.

Returns

One of the following values:

- If successful, returns the pointer to the name of the first attribute in an entry. When you are done using this data, you should free the memory by calling the `ldap_memfree()` function.
- If unsuccessful, returns a `NULL` and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function. (See Chapter 19, "Result Codes for a complete listing of error codes.)

Example

The following section of code retrieves each attribute for an entry.

Code Example 18-14 ldap_first_attribute() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)"
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, \
  my_filter, NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Retrieve the attributes the entry */
  for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
   a = ldap_next_attribute( ld, e, ber ) ) {
    ...
    /* Code to get and manipulate attribute values */
    ...
```

Code Example 18-14 ldap_first_attribute() code example

```
#include <stdio.h>
    }
    ldap_memfree( a );
  }
  /* Free the BerElement from memory when done */
  if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
  }
...
```

See Also

`ldap_first_entry(), ldap_next_entry(), ldap_next_attribute().`

ldap_first_entry()

Returns a pointer to the `LDAPMessage` structure representing the first directory entry in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_first_entry( LDAP *ld, LDAPMessage *result );
```

Parameters

This function has the following parameters:

Table 18-50 ldap_first_entry() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`result`	Chain of search results, which are represented by a pointer to the `LDAPMessage` structure.

Returns

One of the following values:

- If successful, returns the pointer to the first `LDAPMessage` structure of the type `LDAP_RES_SEARCH_ENTRY` in the chain of search results.
- If no `LDAPMessage` structures of the type `LDAP_RES_SEARCH_ENTRY` are in the chain of the search results or if the function is unsuccessful, returns a `NULLMSG`.

Description

The `ldap_first_entry()` function returns a pointer to the `LDAPMessage` structure representing the first directory entry in a chain of search results. Search result entries are in messages of the type `LDAP_RES_SEARCH_ENTRY`.

You can use this function in conjunction with the `ldap_next_entry()` function to iterate through the directory entries in a chain of search results. These functions skip over any messages in the chain that do not have the type `LDAP_RES_SEARCH_ENTRY`.

Do not free the `LDAPMessage` structure returned by this function. Because this is a structure within a chain of search results, freeing this structure will free part of the chain of search results. When you are done working with the search results, you can free the chain itself, rather than individual structures within the chain.

For more information, see “Iterating Through a Chain of Results.”

See Also

`ldap_result()`, `ldap_search_ext()`, `ldap_search_ext_s()`, `ldap_next_entry()`.

ldap_first_message()

Returns a pointer to the first `LDAPMessage` structure in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_first_message( LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the following parameters:

Table 18-51 ldap_first_message() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`res`	Chain of search results, represented by a pointer to the `LDAPMessage` structure.

Returns

One of the following values:

- If successful, returns the pointer to the first LDAPMessage structure in the chain of search results.
- If no LDAPMessage structures are in the chain or if the function is unsuccessful, returns a NULLMSG.

Description

The ldap_first_message() function returns a pointer to the first LDAPMessage structure in a chain of search results.

You can use this function in conjunction with the ldap_next_message() function to iterate through the chain of search results. You can call the ldap_msgtype() function to determine if each message contains a matching entry (a message of the type LDAP_RES_SEARCH_ENTRY) or a search reference (a message of the type LDAP_RES_SEARCH_REFERENCE).

Do not free the LDAPMessage structure returned by this function. Because this is the first structure within a chain of search results, freeing this structure will free the chain of search results. When you are done working with the search results, you can free the chain itself, rather than individual structures within the chain.

For more information, see “Iterating Through a Chain of Results.”

Example

See the examples under “Example: Searching the Directory (Asynchronous)” and “Example: Searching the Directory (Synchronous).”

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(), ldap_next_message().

ldap_first_reference()

Returns a pointer to the LDAPMessage structure representing the first search reference in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_first_reference(LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the following parameters:

Table 18-52 ldap_first_reference() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`res`	Chain of search results, which are represented by a pointer to the `LDAPMessage` structure.

Returns

One of the following values:

- If successful, returns the pointer to the first `LDAPMessage` structure of the type `LDAP_RES_SEARCH_REFERENCE` in the chain of search results.
- If no `LDAPMessage` structures of the type `LDAP_RES_SEARCH_REFERENCE` are in the chain of the search results or if the function is unsuccessful, returns a `NULLMSG`.

Description

The `ldap_first_reference()` function returns a pointer to the `LDAPMessage` structure representing the first search reference in a chain of search results. Search references are in messages of the type `LDAP_RES_SEARCH_REFERENCE`.

You can use this function in conjunction with the `ldap_next_reference()` function to iterate through the search references in a chain of search results. These functions skip over any messages in the chain that do not have the type `LDAP_RES_SEARCH_REFERENCE`.

Do not free the `LDAPMessage` structure returned by this function. Because this is a structure within a chain of search results, freeing this structure will free part of the chain of search results. When you are done working with the search results, you can free the chain itself, rather than individual structures within the chain.

For more information, see “Iterating Through a Chain of Results.”

See Also

`ldap_result()`, `ldap_search_ext()`, `ldap_search_ext_s()`, `ldap_next_reference()`.

ldap_free_friendlymap()

The `ldap_free_friendlymap()` function frees the structures allocated by the `ldap_friendly_name()` function.

Syntax

```
#include <ldap.h>
void ldap_free_friendlymap( FriendlyMap *map );
```

Parameters

This function has the following parameters:

Table 18-53 ldap_free_friendlymap() function parameters

`map`	Pointer to the mapping structure in memory.

Example

The following section of code frees memory allocated by the `ldap_friendly_name()` function.

Code Example 18-15 ldap_free_friendlymap() code example

```
#include <ldap.h>
...
FriendlyMap *map;
...
ldap_free_friendlymap( map );
...
```

See Also

`ldap_friendly_name()`.

ldap_free_sort_keylist()

Frees the structures allocated by the `ldap_create_sort_keylist()` function.

Syntax

```
#include <ldap.h>
void ldap_free_sort_keylist (LDAPsortkey **sortKeyList);
```

Parameters

This function has the following parameters:

Table 18-54 ldap_free_sort_keylist() function parameters

`sortKeyList`	Array of `LDAPsortkey` structures that you want to free from memory.

Description

The `ldap_free_sort_keylist()` function frees the array of `LDAPsortkey` structures allocated by the `ldap_create_sort_keylist()` function.

When you are done sorting results, you can call this function to free the memory that you have allocated.

See Also

`ldap_create_sort_keylist()`.

ldap_free_urldesc()

The `ldap_free_urldesc()` function frees memory allocated by the `ldap_url_parse()` function. For more information, see "Freeing the Components of an LDAP URL."

Syntax

```
#include <ldap.h>
void ldap_free_urldesc( LDAPURLDesc *ludp );
```

Parameters

This function has the following parameters:

Table 18-55 ldap_free_urldesc() function parameters

`ludp`	Pointer to a `LDAPURLDesc` structure.

Example

The following section of code parses an LDAP URL and then frees the `LDAPURLDesc` structure from memory after verifying that the LDAP URL is valid.

Code Example 18-16 ldap_free_urldesc() code example

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url =
"ldap://ldap.netscape.com:5000/o=airius.com?cn,mail,telephoneNumber?sub? \
  (sn=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
  switch( res ){
    case LDAP_URL_ERR_NOTLDAP:
      printf( "URL does not begin with \"ldap://\"\n" );
      break;
    case LDAP_URL_ERR_NODN:
      printf( "URL does not contain a distinguished name\n" );
      break;
    case LDAP_URL_ERR_BADSCOPE:
      printf( "URL contains an invalid scope\n" );
      break;
    case LDAP_URL_ERR_MEM:
      printf( "Not enough memory\n" );
      break;
    default:
      printf( "Unknown error\n" );
  }
  return( 1 );
}
printf( "URL is a valid LDAP URL\n" );
ldap_free_urldesc( ludpp );
...
```

See Also
`ldap_url_parse()`.

ldap_friendly_name()

The `ldap_friendly_name()` function maps a set of "unfriendly names" (for example, a two-letter country code such as "`IS`") to "friendly names" (for example, the full names of countries, such as "`Iceland`").

This function relies on the existence of a file mapping "unfriendly names" to "friendly names". The names in the file are tab-delimited, as shown in the example file below:

```
unfriendly_name>        <friendly_name>
AD        Andorra
AE        United Arab Emirates
AF        Afghanistan
AG        Antigua and Barbuda
AI        Anguilla
```

Syntax

```
#include <ldap.h>
char * ldap_friendly_name( char *filename, char *uname,
  FriendlyMap *map );
```

Parameters

This function has the following parameters:

Table 18-56 ldap_friendly_name() function parameters

`filename`	Name of the map file listing the "unfriendly names" and "friendly names."
`uname`	"Unfriendly name" for which you want to find the "friendly name."
`map`	Pointer to the mapping in memory. Initialize this pointer to `NULL` on the first call, then use the pointer during subsequent calls so that the mapping file does not need to be read again.

Returns

One of the following values:

- If successful, returns the "friendly name" for the specified "unfriendly name".
- If unsuccessful (for example, if the file cannot be read, if the file is in a bad format, or if the map parameter is set to `NULL`), returns the original name (the name you passed as the `uname` parameter).

Example

The following section of code reads in a map of friendly names and prints the name corresponding to the unfriendly name "IS".

Code Example 18-17 ldap_friendly_name() code example

```
#include <ldap.h>
#include <stdio.h>
...
FriendlyMap map = NULL;
char *map_file = "/u/mozilla/ldapsdk/examples/ldapfriendly";
char *unfriendly_name = "IS";
```

Code Example 18-17 ldap_friendly_name() code example

```
#include <ldap.h>
char *friendly_name;
...
/* Read the ldapfriendly file into the map in memory */
friendly_name = ldap_friendly_name( map_file, unfriendly_name, &map );
printf( "Friendly Name for %s: %s\n", unfriendly_name, friendly_name );

/* Since the file is in memory now, no need to reference it in subsequent calls
*/
friendly_name = ldap_friendly_name( NULL, "VI", &map );
printf( "Friendly Name for VI: %s\n", friendly_name );
...
```

See Also

`ldap_free_friendlymap()`.

ldap_get_dn()

The `ldap_get_dn()` routine returns the distinguished name (DN) for an entry in a chain of search results. You can get an entry from a chain of search results by calling the `ldap_first_entry()` and `ldap_next_entry()` functions. For more information, see "Getting Distinguished Names for Each Entry."

Syntax

```
#include <ldap.h>
char * ldap_get_dn( LDAP *ld, LDAPMessage *entry );
```

Parameters

This function has the following parameters:

Table 18-57 ldap_get_dn() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`entry`	Pointer to an entry in a chain of search results, as returned by the `ldap_first_entry()` and `ldap_next_entry()` functions.

Returns

One of the following values:

- If successful, returns the distinguished name (DN) for the specified entry.
- If unsuccessful, returns a `NULL` and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function. (See Chapter 19, “Result Codes” for a complete listing of error codes.)

Example

The following section of code prints the distinguished name for each entry found in a search.

Code Example 18-18 ldap_get_dn() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
char *dn;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
  NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* For each matching entry found, print out the name of the entry.*/
for ( e = ldap_first_entry( ld, result ); e != NULL;
  e = ldap_next_entry( ld, e ) ) {
  if ( ( dn = ldap_get_dn( ld, e ) ) != NULL ) {
    printf( "dn: %s\n", dn );
    /* Free the memory used for the DN when done */
    ldap_memfree( dn );
  }
}
/* Free the result from memory when done. */
ldap_msgfree( result );
...
```

See Also

`ldap_first_entry()`, `ldap_next_entry()`.

ldap_get_entry_controls()

Gets the LDAP controls included with a directory entry in a set of search results.

Syntax

```
#include <ldap.h>
int ldap_get_entry_controls( LDAP *ld, LDAPMessage *entry,
  LDAPControl ***serverctrlsp );
```

Parameters

This function has the following parameters:

Table 18-58 ldap_get_entry_controls() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`entry`	Pointer to an `LDAPMessage` structure representing an entry in a chain of search results.
`serverctrlsp`	Pointer to an array of `LDAPControl` structures, which represent the LDAPv3 server controls returned by the server.

Returns

One of the following values:

- `LDAP_SUCCESS` if the LDAP controls were successfully retrieved.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded message.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NO_MEMORY` if memory cannot be allocated.

Description

The `ldap_get_entry_controls()` function retrieves the LDAPv3 controls included in a directory entry in a chain of search results.

The LDAP controls are specified in an array of `LDAPControl` structures. (Each `LDAPControl` structure represents an LDAP control.)

At this point in time, the entry notification controls (which are used with persistent search controls) are the only controls that are returned with individual entries. Other controls are returned with results sent from the server. You can call `ldap_parse_result()` to retrieve those controls.

ldap_getfilter_free()

The `ldap_getfilter_free()` function frees the memory used by a filter set. Once you call this routine, the `LDAPFiltDesc` structure is no longer valid and cannot be used again. For more information, see "Freeing Filters from Memory."

Syntax

```
#include <ldap.h>
void ldap_getfilter_free( LDAPFiltDesc *lfdp );
```

Parameters

This function has the following parameters:

Table 18-59 ldap_getfilter_free() function parameters

`lfdp`	Pointer to a `LDAPFiltDesc` structure.

Example

The following section of code frees the `LDAPFiltDesc` structure from memory after all searches are completed.

Code Example 18-19 ldap_getfilter_free code example

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
...
/* Read the filter configuration file into an LDAPFiltDesc structure */
lfdp = ldap_init_getfilter( filter_file );
...
/* Retrieve filters and perform searches */
...
/* Free the configuration file (the LDAPFiltDesc structure) */
ldap_getfilter_free( lfdp );
...
```

See Also

`ldap_init_getfilter()`, `ldap_init_getfilter_buf()`.

ldap_getfirstfilter()

The `ldap_getfirstfilter()` function retrieves the first filter that is appropriate for a given value. For more information, see “Retrieving Filters.”

Syntax

```
#include <ldap.h>
LDAPFiltInfo * ldap_getfirstfilter( LDAPFiltDesc *lfdp,
  char *tagpat, char *value );
```

Parameters

This function has the following parameters:

Table 18-60 ldap_getfirstfilter() function parameters

`lfdp`	Pointer to an `LDAPFiltDesc` structure.
`tagpat`	Regular expression for a tag in the filter configuration.
`value`	Value for which to find the first appropriate filter.

Returns

One of the following values:

- If successful, returns a pointer to an `LDAPFiltInfo` structure.
- If no more filters can be returned, returns a `NULL`.
- If unsuccessful, returns a `NULL`.

Example

The following section of code is based on the `getfilt` command-line program example provided with the LDAP SDK for C. The program prompts the user to enter search criteria. Based on the criteria entered, the program retrieves filters that match the search criteria. The example uses the filter configuration file shown in “Understanding the Configuration File Syntax.”

Code Example 18-20 ldap_getfirstfilter() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a, *dn;
char **vals;
```

Code Example 18-20 ldap_getfirstfilter() code example

```
int i;
LDAPFiltDesc *ldfp;
LDAPFiltInfo *ldfi;
char buf[ 80 ]; /* Contains the search criteria */
int found;
...
/* Load the filter configuration file into an LDAPFiltDesc structure */
if ( ( ldfp = ldap_init_getfilter( "myfilters.conf" ) ) == NULL ) {
  perror( "Cannot open filter configuration file" );
  return( 1 );
}

/* Read a string to search for */
printf( "Enter a string to search for: " );
gets( buf );
if ( strlen( buf ) == 0 ) {
  fprintf( stderr, "usage: %s search-string\n", argv[ 0 ]);
  return( 1 );
}

/* Select a filter to use when searching for the value in buf */
found = 0;
for ( ldfi = ldap_getfirstfilter( ldfp, "people", buf ); ldfi != NULL; ldfi =
ldap_getnextfilter( ldfp ) ) {

  /* Use the selected filter to search the directory */
  if ( ldap_search_s( ld, "o=airius.com, ldfi->lfi_scope,
   ldfi->lfi_filter, NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
  } else {

    /* Once a filter gets results back, stop iterating through the different
filters */
    if ( ( found = ldap_count_entries( ld, result ) > 0 ) ) {
      break;
    } else {
      ldap_msgfree( result );
    }
  }
}

if ( found == 0 ) {
  printf( "No matching entries found.\n" );
} else {
  printf( "Found %d %s match%s for \"%s\"\n\n", found,
   ldfi->lfi_desc, found == 1 ? "" : "es", buf );
}

ldap_msgfree( result );
ldap_getfilter_free( ldfp );
...
```

See Also

ldap_init_getfilter(), ldap_init_getfilter_buf(), ldap_getnextfilter().

ldap_get_lang_values()

The ldap_get_lang_values() function returns a NULL-terminated array of an attribute's string values that match a specified language subtype.

If you want to retrieve binary data from an attribute, call the ldap_get_lang_values_len() function instead.

Syntax

```
#include <ldap.h>
char ** ldap_get_lang_values( LDAP *ld, LDAPMessage *entry,
  const char *target, char **type );
```

Parameters

This function has the following parameters:

Table 18-61 ldap_get_lang_values() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Entry retrieved from the directory.
target	Attribute type (including an optional language subtype) that you want to retrieve the values of.
type	Pointer to a buffer that returns the attribute type retrieved by this function.

Returns

One of the following values:

- If successful, returns a NULL-terminated array of the attribute's values.
- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the ldap_get_lderrno() function. (See Chapter 19, "Result Codes" for a complete listing of error codes.)

Description

Unlike the ldap_get_values() function, if a language subtype is specified, this function first attempts to find and return values that match that subtype (for example, cn;lang-en).

See Also

ldap_first_entry(), ldap_next_entry(), ldap_first_attribute(), ldap_next_attribute(), ldap_get_lang_values_len(), ldap_get_values().

ldap_get_lang_values_len()

The ldap_get_lang_values_len() function returns a NULL-terminated array of pointers to berval structures, each containing the length and pointer to a binary value of an attribute for a given entry. Use the ldap_get_lang_values() routine instead of this routine if the attribute values are string values.

For more information, see "Getting the Values of an Attribute."

Syntax

```
#include <ldap.h>
struct berval ** ldap_get_lang_values_len( LDAP *ld,
  LDAPMessage *entry, const char *target, char **type );
```

Parameters

This function has the following parameters:

Table 18-62 ldap_get_lang_values_len() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Result returned by the ldap_result() or ldap_search_s() function.
target	Attribute returned by ldap_first_attribute() or ldap_next_attribute(), or the attribute as a literal string, such as "jpegPhoto" or "audio".
type	Pointer to a buffer that returns the attribute type retrieved by this function.

Returns

One of the following values:

- If successful, returns a NULL-terminated array of pointers to berval structures, which in turn contain pointers to the attribute's binary values.

- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function. (See Chapter 19, “Result Codes” for a complete listing of error codes.)

See Also

`ldap_first_entry(), ldap_next_entry(), ldap_first_attribute(), ldap_next_attribute(), ldap_get_lang_values(), ldap_get_values_len().`

ldap_get_lderrno()

The `ldap_get_lderrno()` function gets information about the last error that occurred for an LDAP operation. You can also call this function to get error codes for functions that do not return errors (such as `ldap_next_attribute()`).

For more information, see “Getting Information About the Error.”

Syntax

```
#include <ldap.h>
int ldap_get_lderrno( LDAP *ld, char **m, char **s );
```

Parameters

This function has the following parameters:

Table 18-63 ldap_get_lderrno() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`m`	In the event of an `LDAP_NO_SUCH_OBJECT` error return, this parameter contains the portion of the DN that identifies an existing entry. (See “Receiving the Portion of the DN Matching an Entry.”)
`s`	The text of the error message.

Returns

The LDAP error code for the last operation. (See Chapter 19, “Result Codes” for a complete listing of error codes.)

Example

The following section of code attempts to add a new user to the directory. If the entry identified by a DN does not exist, the server sends the client a portion of the DN that find an existing entry. This DN is returned to the client as the variable `matched`. (See "Receiving the Portion of the DN Matching an Entry" for details.)

Code Example 18-21 ldap_get_lderrno() code example

```
#include <ldap.h>
LDAP *ld;
char *dn = "uid=bjensen, ou=People, o=airius.com";
LDAPMod **list_of_attrs;
char *matched;
int rc;
...
if ( ldap_add_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
  rc = ldap_get_lderrno( ld, &matched, NULL );
  return( rc );
}
...
```

In the example above, if no organizational unit named New Department exists, the `matched` variable is set to:

```
o=airius.com
```

See Also

`ldap_err2string()`, `ldap_perror()`, `ldap_result2error()`, `ldap_set_lderrno()`.

ldap_getnextfilter()

The `ldap_getnextfilter()` function retrieves the next filter that is appropriate for a given value. Call this function to get subsequent filters after calling `ldap_getfirstfilter()`. For more information, see "Retrieving Filters."

Syntax

```
#include <ldap.h>
LDAPFiltInfo * ldap_getnextfilter( LDAPFiltDesc *lfdp );
```

Parameters

This function has the following parameters:

Table 18-64 ldap_getnextfilter() function parameters

`lfdp`	Pointer to an `LDAPFiltDesc` structure.

Returns

One of the following values:

- If successful, returns a pointer to an `LDAPFiltInfo` structure.
- If no more filters can be returned, returns a `NULL`.
- If unsuccessful, returns a `NULL`.

Example

See the example under `ldap_getfirstfilter()`.

See Also

`ldap_getfirstfilter()`.

ldap_get_option()

The function `ldap_get_option()` gets session preferences from an `LDAP` structure. For information on the options you can retrieve with this function, see `ldap_set_option()`.

Syntax

```
#include <ldap.h>
int ldap_get_option( LDAP *ld, int option, void *optdata );
```

Parameters

This function has the following parameters:

Table 18-65 ldap_get_option() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`option`	Option that you want to retrieve. The option parameter must be set to one of the option values, as detailed in Table 18-117, page 491, which describes the function parameters for `ldap_set_option()`.

Table 18-65 ldap_get_option() function parameters

`optdata`	Pointer to the buffer in which the value of the option will be put.

The following table describes the options that you can retrieve with `ldap_get_option()`.

Table 18-66 Options for the ldap_get_options() function

`LDAP_OPT_API_FEATURE_INFO`	Retrieves information about the revision of a supported LDAP feature.
`LDAP_OPT_API_INFO`	Retrieves information the API and the extensions supported, including the supported API version, protocol version, the names of the supported API extensions with their vendor name version. For details on the structure returned, refer to the `ldap.h` header file.
`LDAP_OPT_CLIENT_CONTROLS`	Pointer to an array of `LDAPControl` structures representing the LDAPv3 client controls you want sent with every request by default. The data type for the `optdata` parameter is `(LDAPControl ***)`.
`LDAP_OPT_DESC`	Socket descriptor underlying the main LDAP connection. The data type for the `optdata` parameter is `(LBER_SOCKET *)`. The `LBER_SOCKET` data type depends on the platform that you are using: • `int` in UNIX. • `SOCKET` in Windows.

Table 18-66 Options for the ldap_get_options() function

LDAP_OPT_DEREF	Determines how aliases work during a search. The data type for the optdata parameter is (int *). optdata can be one of the following values: • LDAP_DEREF_NEVER specifies that aliases are never dereferenced. • LDAP_DEREF_SEARCHING specifies that aliases are dereferenced when searching under the base object (but not when finding the base object). • LDAP_DEREF_FINDING specifies that aliases are dereferenced when finding the base object (but not when searching under the base object). • LDAP_DEREF_ALWAYS specifies that aliases are always dereferenced when finding and searching under the base object.
LDAP_OPT_DNS_FN_PTRS	Lets you use alternate DNS functions for getting the host entry of the LDAP server. The data type for the optdata parameter is (struct ldap_dns_fns *).
LDAP_OPT_ERROR_NUMBER	Result code for the most recent LDAP error that occurred for this session. The data type for the optdata parameter is (int *).
LDAP_OPT_ERROR_STRING	Error message returned with the result code for the most recent LDAP error that occurred for this session. The data type for the optdata parameter is (char **).
LDAP_OPT_EXTRA_THREAD_FN_PTRS	Lets you specify the locking and semaphore functions that you want called when getting results from the server. (See Chapter 16, “Writing Multithreaded Clients” for details.) The data type for the optdata parameter is (struct ldap_extra_thread_fns *).
LDAP_OPT_IO_FN_PTRS	Lets you use alternate communication stacks. The data type for the optdata parameter is (struct ldap_io_fns *).
LDAP_OPT_MATCHED_DN	Gets the matched DN value returned with the most recent LDAP error that occurred for this session.

Table 18-66 Options for the ldap_get_options() function

LDAP_OPT_MEMALLOC_FN_PTRS	Gets a pointer to the callback structure which you previously set. The data type for the optdata parameter is (struct ldap_memalloc_fnsldap_io_fns *).
LDAP_OPT_PROTOCOL_VERSION	Version of the protocol supported by your client. The data type for the optdata parameter is (int *). You can specify either LDAP_VERSION2 or LDAP_VERSION3. If no version is set, the default is LDAP_VERSION2. In order to use LDAPv3 features, you need to set the protocol version to LDAP_VERSION3.
LDAP_OPT_REBIND_ARG	Lets you set the last argument passed to the routine specified by LDAP_OPT_REBIND_FN. You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the optdata parameter is (void *).
LDAP_OPT_REBIND_FN	Lets you set the routine to be called when you need to authenticate a connection with another LDAP server (for example, during the course of following a referral). You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the optdata parameter is (LDAP_REBINDPROC_CALLBACK *).
LDAP_OPT_RECONNECT	If the connection to the server is lost, determines whether or not the same connection handle should be used to reconnect to the server. The data type for the optdata parameter is (int *). optdata can be one of the following values: • LDAP_OPT_ON specifies that the same connection handle can be used to reconnect to the server. • LDAP_OPT_OFF specifies that you want to create a new connection handle to connect to the server. By default, this option is off. For details, see "Handling Failover."

Table 18-66 Options for the ldap_get_options() function

LDAP_OPT_REFERRALS	Determines whether or not the client should follow referrals. The data type for the `optdata` parameter is `(int *)`. `optdata` can be one of the following values: • `LDAP_OPT_ON` specifies that the server should follow referrals. • `LDAP_OPT_OFF` specifies that the server should not follow referrals. By default, the client follows referrals.
LDAP_OPT_REFERRAL_HOP_LIMIT	Maximum number of referrals the client should follow in a sequence (in other words, the client can only be referred this number of times before it gives up). The data type for the `optdata` parameter is `(int *)`. By default, the maximum number of referrals that the client can follow in a sequence is 5 for the initial connection. Note that this limit does not apply to individual requests that generate multiple referrals in parallel.
LDAP_OPT_RESTART	Determines whether or not LDAP I/O operations should be restarted automatically if they are prematurely aborted. The data type for the `optdata` parameter is `(int *)`. `optdata` can be one of the following values: • `LDAP_OPT_ON` specifies that I/O operations should be restarted automatically. • `LDAP_OPT_OFF` specifies that I/O operations should not be restarted automatically.
LDAP_OPT_SERVER_CONTROLS	Pointer to an array of `LDAPControl` structures representing the LDAPv3 server controls you want sent with every request by default. Typically, since controls are specific to the type of request, you may want to pass the controls using operation-specific functions (such as `ldap_add_ext()`) instead. The data type for the `optdata` parameter is `(LDAPControl ***)`.

Table 18-66 Options for the ldap_get_options() function

LDAP_OPT_SIZELIMIT	Maximum number of entries that should be returned by the server in search results. The data type for the `optdata` parameter is `(int *)`. Note that the LDAP server may impose a smaller size limit than the limit you specify. (The server administrator has the ability to set this limit.)
LDAP_OPT_SSL	Determines whether or not SSL is enabled. The data type for the `optdata` parameter is `(int *)`. `optdata` can be one of the following values: • LDAP_OPT_ON specifies that SSL is enabled. • LDAP_OPT_OFF specifies that SSL is disabled.
LDAP_OPT_THREAD_FN_PTRS	Lets you specify the thread function pointers. (See Chapter 16, "Writing Multithreaded Clients" for details.) The data type for the `optdata` parameter is `(struct ldap_thread_fns *)`.
LDAP_OPT_TIMELIMIT	Maximum number of seconds that should be spent by the server when answering a search request. The data type for the `optdata` parameter is `(int *)`. Note that the LDAP server may impose a shorter time limit than the limit you specify. (The server administrator has the ability to set this limit.)

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- -1 if unsuccessful.

Examples

The following example gets the session preference for the maximum number of entries to be returned from search operations.

Code Example 18-22 ldap_get_option() code example

```
#include <ldap.h>
...
LDAP *ld;
int max_ret;
...
/* Get the maximum number of entries returned */
if (ldap_get_option, LDAP_OPT_SIZELIMIT, &max_ret) != LDAP_SUCCESS) {
  ldap_perror( ld, "ldap_get_option" );
  return( 1 );
}
```

Here are two small sections of code that show how to use the `LDAP_OPT_API_FEATURE_INFO` and the `LDAP_OPT_API_INFO` options, respectively:

```
LDAPIIFeatureInfo ldfi;
ldfi.ldapaif_info_version = LDAP_FEATURE_INFO_VERSION;
ldfi.ldapaif_name = "VIRTUAL_LIST_VIEW";
if (ldap_get_option(NULL, LDAP_OPT_API_FEATURE_INFO, &ldfi)==0) {
/* use the info here */ }

LDAPIInfo ldai;
ldai.ldapiai_info_version = LDAP_API_INFO_VERSION;
if (ldap_get_option( NULL, LDAP_OPT_API_INFO, &ldia ) == 0) {
/* use the ldai info here */
}
```

See Also

`ldap_init()`, `ldap_set_option()`.

ldap_get_values()

The `ldap_get_values()` function returns a `NULL`-terminated array of an attribute's string values for a given entry. Use the `ldap_get_values_len()` function instead of this function if the attribute values are binary.

For more information, see "Getting the Values of an Attribute."

Syntax

```
#include <ldap.h>
char ** ldap_get_values( LDAP *ld, LDAPMessage *entry,
  const char *target );
```

Parameters

This function has the following parameters:

Table 18-67 ldap_get_values() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`entry`	Result returned by the `ldap_result()` or `ldap_search_s()` function.
`target`	Attribute returned by `ldap_first_attribute()` or `ldap_next_attribute()`, or the attribute as a literal string, such as "`jpegPhoto`" or "`audio`".

Returns

One of the following values:

- If successful, returns a `NULL`-terminated array of the attribute's values.
- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function. (See Chapter 19, "Result Codes" for a complete listing of error codes.)

Example

The following section of code gets and prints the values of an attribute in an entry. This example assumes that all attributes have string values.

Code Example 18-23 ldap_get_values() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a;
char **vals;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
int i;
...
```

Code Example 18-23 ldap_get_values() code example

```
#include <stdio.h>
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
  NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Get the first matching attribute */
a = ldap_first_attribute( ld, e, &ber );

/* Get the values of the attribute */
if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
  for ( i = 0; vals[i] != NULL; i++ ) {
    /* Print the name of the attribute and each value */
    printf( "%s: %s\n", a, vals[i] );
  }
  /* Free the attribute values from memory when done. */
  ldap_value_free( vals );
}
...
```

See Also

`ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_attribute()`, `ldap_next_attribute()`, `ldap_get_lang_values()`, `ldap_get_values_len()`.

ldap_get_values_len()

The `ldap_get_values_len()` function returns a `NULL`-terminated array of pointers to `berval` structures, each containing the length and pointer to a binary value of an attribute for a given entry. Use the `ldap_get_values()` routine instead of this routine if the attribute values are string values.

For more information, see “Getting the Values of an Attribute.”

Syntax

```
#include <ldap.h>
struct berval ** ldap_get_values_len( LDAP *ld,
  LDAPMessage *entry, const char *target );
```

Parameters

This function has the following parameters:

Table 18-68 ldap_get_values_len() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`entry`	Result returned by the `ldap_result()` or `ldap_search_s()` function.
`target`	Attribute returned by `ldap_first_attribute()` or `ldap_next_attribute()`, or the attribute as a literal string, such as "`jpegPhoto`" or "`audio`".

Returns

One of the following values:

- If successful, returns a `NULL`-terminated array of pointers to `berval` structures, which in turn contain pointers to the attribute's binary values.
- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function. (See Chapter 19, "Result Codes" for a complete listing of error codes.)

Example

The following section of code gets the first value of the `jpegPhoto` attribute and saves the JPEG data to a file.

Code Example 18-24 ldap_get_values_len() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
struct berval photo_data;
struct berval **list_of_photos;
FILE *out;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter, NULL,
  0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
```

Code Example 18-24 ldap_get_values_len() code example

```
#include <stdio.h>
  return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Get the value of the jpegPhoto attribute */
if ( ( list_of_photos = ldap_get_values_len( ld, e, "jpegPhoto" ) ) != NULL ) {
  /* Prepare to write the JPEG data to a file */
  if ( ( out = fopen( "photo.jpg", "wb" ) ) == NULL ) {
    perror( "fopen" );
    return( 1 );
  }
  /* Get the first JPEG */
  photo_data = *list_of_photos[0];
  /* Write the JPEG data to a file */
  fwrite( photo_data.bv_val, photo_data.bv_len, 1, out );
  fclose( out );
  /* Free the attribute values from memory when done. */
  ldap_value_free_len( list_of_photos );
}
...
```

See Also

ldap_first_entry(), ldap_next_entry(), ldap_first_attribute(), ldap_next_attribute(), ldap_get_lang_values_len(), ldap_get_values().

ldap_init()

Initializes a session with an LDAP server and returns an LDAP structure that represents the context of the connection to that server.

Syntax

```
#include <ldap.h>
LDAP * ldap_init( const char *defhost, int defport );
```

Parameters

This function has the following parameters:

Table 18-69 ldap_init() function parameters

`defhost`	Space-delimited list of one or more host names (or IP address in dotted notation, such as "141.211.83.36") of the LDAP servers that you want the LDAP client to connect to. The names can be in *hostname:portnumber* format (in which case, *portnumber* overrides the port number specified by the `defport` argument.
`defport`	Default port number of the LDAP server. To specify the standard LDAP port (port 389), use `LDAP_PORT` as the value for this parameter.

Returns

One of the following values:

- If successful, returns a pointer to an LDAP structure.
- If unsuccessful, returns a `NULL`.

Description

The `ldap_init()` function initializes a session with an LDAP server. `ldap_init()` allocates an LDAP structure containing information about the session, including the host name and port of the LDAP server, preferences for the session (such as the maximum number of entries to return in a search), and the error code of the last LDAP operation performed.

You can specify a list of LDAP servers that you want to attempt to connect to. Your client will attempt to connect to the first LDAP server in the list. If the attempt fails, your client will attempt to connect to the next LDAP server in the list.

You can specify the list of LDAP servers by passing a space-delimited list of the host names as the host argument. For example:

```
...
LDAP *ld
...
ld = ldap_init( "ld1.netscape.com ld2.netscape.com \
  ld3.netscape.com", LDAP_PORT );
```

In the example above, the LDAP client will attempt to connect to the LDAP server on `ld1.netscape.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.netscape.com`, port 389. If that server does not respond, the client will use the server on `ld3.netscape.com`, port 389.

If any of the servers do not use the default port specified by the `defport` argument, use the *host:port* format to specify the server name. For example:

```
...
LDAP *ld
...
ld = ldap_init( "ld1.netscape.com ld2.netscape.com:38900", \
  LDAP_PORT );
```

In the example above, the LDAP client will attempt to connect to the LDAP server on `ld1.netscape.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.netscape.com`, port 38900.

Once you initialize a session, you need to call the `ldap_simple_bind()` or `ldap_simple_bind_s()` function to connect and authenticate to the LDAP server.

For more information, see “Initializing an LDAP Session.”

Note that if you are connecting to a secure LDAP server over SSL, you should be calling the `ldapssl_init()` function instead. For details, see Chapter 12, “Connecting Over SSL.”

Example

The following section of code initializes a session with the LDAP server at `ldap.netscape.com:389`.

Code Example 18-25 ldap_init() code example

```
#include <ldap.h>
...
LDAP *ld;

/* Specify the host name of the LDAP server. */
char *ldap_host = "ldap.netscape.com";

/* Because the LDAP server is running on the standard LDAP port (port 389), you
can use LDAP_PORT to identify the port number. */
int ldap_port = LDAP_PORT;
...
/* Initialize the session with ldap.netscape.com:389 */
if ( ( ld = ldap_init( ldap_host, ldap_port ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
```

Code Example 18-25 ldap_init() code example

```
}
...
/* Subsequent calls that authenticate to the LDAP server. */
...
```

See Also

`ldap_unbind(), ldap_unbind_s().`

ldap_init_getfilter()

The `ldap_init_getfilter()` function reads a valid LDAP filter configuration file (such as `ldapfilter.conf`) and returns a pointer to an `LDAPFiltDesc` structure.

For more information, see "Loading a Filter Configuration File."

Syntax

```
#include <ldap.h>
LDAPFiltDesc * ldap_init_getfilter( char *fname );
```

Parameters

This function has the following parameters:

Table 18-70 ldap_init_getfilter() function parameters

`fname`	Name of the LDAP filter configuration file to use.

Returns

One of the following values:

- If successful, returns a pointer to an `LDAPFiltDesc` structure.
- If unsuccessful (for example, if there is an error reading the file), returns a `NULL`.

Example

The following section of code loads the filter configuration file named `myfilters.conf` into memory.

Code Example 18-26 Loading a filter configuration file

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
...
lfdp = ldap_init_getfilter( filter_file );
...
```

See Also

`ldap_init_getfilter_buf(), ldap_getfilter_free().`

ldap_init_getfilter_buf()

The `ldap_init_getfilter_buf()` function reads LDAP filter configuration information from a buffer and returns a pointer to an `LDAPFiltDesc` structure. The configuration information in the buffer must use the correct syntax, as specified in "Understanding the Configuration File Syntax."

For more information, see "Loading a Filter Configuration File."

Syntax

```
#include <ldap.h>
LDAPFiltDesc * ldap_init_getfilter_buf(char *buf, long buflen );
```

Parameters

This function has the following parameters:

Table 18-71 ldap_init_getfilter_buf() function parameters

`buf`	Buffer containing LDAP filter configuration information.
`buflen`	Size of the buffer

Returns

One of the following values:

- If successful, returns a pointer to an `LDAPFiltDesc` structure.
- If unsuccessful (for example, if there is an error reading the file), returns a `NULL`.

Example

The following section of code copies the following filter configuration to a buffer in memory.

```
"ldap-example"
  "@"      " "    "(mail=%v)"            "email address"
            "(mail=%v*)"           "start of email address"
```

The example uses this buffer to fill an `LDAPFiltDesc` structure.

Code Example 18-27 ldap_init_getfilter() code example

```
#include <string.h>
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char filtbuf[ 1024 ];
...
/* Create the filter config buffer */
strcpy( filtbuf, "\"ldap-example\"\n" );
strcat( filtbuf, " \"@\"\t\" \"\t\"(mail=%v)\"\t\"email address\"\n" );
strcat( filtbuf, " \t\t\"(mail=%v*)\"\t\"start of email address\"\n" );
lfdp = ldap_init_getfilter( filtbuf, strlen( filtbuf ) );
...
```

See Also

`ldap_init_getfilter()`, `ldap_getfilter_free()`.

ldap_is_ldap_url()

The `ldap_is_ldap_url()` function determines whether or not a URL is an LDAP URL. An LDAP URL is a URL with the protocol set to `ldap://` (or `ldaps://`, if the server is communicating over an SSL connection).

For more information, see “Determining If a URL is an LDAP URL.”

Syntax

```
#include <ldap.h>
int ldap_is_ldap_url( const char *url );
```

Parameters

This function has the following parameters:

Table 18-72 ldap_is_ldap_url() function parameters

`url`	The URL that you want to check.

Returns

One of the following values:

- `1` if the URL is an LDAP URL.
- `0` if the URL is not an LDAP URL.

Example

The following section of code determines if a URL is an LDAP URL.

Code Example 18-28 ldap_is_ldap_url() code example

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url = "ldap://ldap.netscape.com/o=airius.com";
...
if ( ldap_is_ldap_url( my_url ) != 0 ) {
  printf( "%s is an LDAP URL.\n", my_url );
} else {
  printf( "%s is not an LDAP URL.\n", my_url );
}
...
```

See Also

`ldap_url_parse()`.

ldap_memcache_destroy()

Frees an `LDAPMemCache` structure from memory.

Syntax

```
#include <ldap.h>
void ldap_memcache_destroy( LDAPMemCache *cache );
```

Parameters

This function has the following parameters:

Table 18-73 ldap_memcache_destroy() function parameters

cache	Pointer to the LDAPMemCache structure that you want freed from memory.

Description

The ldap_memcache_destroy() function frees the specified LDAPMemCache structure from memory. Call this function after you are done working with a cache.

See Also

ldap_memcache_init().

ldap_memcache_flush()

Flushes items from the specified cache.

Syntax

```
#include <ldap.h>
void ldap_memcache_flush( LDAPMemCache *cache, char *dn,
  int scope );
```

Parameters

This function has the following parameters:

Table 18-74 ldap_memcache_flush() function parameters

cache	Pointer to the LDAPMemCache structure that you want to flush entries from.
dn	Base DN identifying the search requests that you want flushed from the cache. If the base DN of a search request is within the scope specified by this DN and the scope argument, the search request is flushed from the cache. If this argument is NULL, the entire cache is flushed.

Table 18-74 ldap_memcache_flush() function parameters

`scope`	Scope that (together with the `dn` argument) identifies the search requests that you want flushed from the cache. If the base DN of the request is within the scope specified by this argument and the `dn` argument, the request is flushed from the cache. This argument can have one of the following values: • `LDAP_SCOPE_BASE` • `LDAP_SCOPE_ONELEVEL` • `LDAP_SCOPE_SUBTREE`

Description

The `ldap_memcache_flush()` function flushes search requests from the cache. If the base DN of a search request is within the scope specified by the `dn` and `scope` arguments, the search request is flushed from the cache.

If no DN is specified, the entire cache is flushed.

ldap_memcache_get()

Gets the in-memory cache associated with an LDAP connection handle.

Syntax

```
#include <ldap.h>
int ldap_memcache_get( LDAP *ld, LDAPMemCache **cachep );
```

Parameters

This function has the following parameters:

Table 18-75 ldap_memcache_get() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`cachep`	When you call this function, the function sets this parameter to the pointer to the `LDAPMemCache` structure associated with the connection handle.

Returns

One of the following values:

- `LDAP_SUCCESS` if the cache for the specified connection handle was retrieved successfully.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.

Description

The `ldap_memcache_get()` function gets the cache associated with the specified connection handle (LDAP structure). This cache is used by all search requests made through that connection.

You can call this function if you want to associate a cache with multiple LDAP connection handles. For example, you can call this function to get the cache associated with one connection, then you can call the `ldap_memcache_set()` function to associate the cache with another connection.

See Also

`ldap_memcache_set()`.

ldap_memcache_init()

Creates an in-memory cache for your LDAP client that you can associate with an LDAP connection.

Syntax

```
#include <ldap.h>
int ldap_memcache_init( unsigned long ttl, unsigned long size,
  char **baseDNs, struct ldap_thread_fns *thread_fns,
  LDAPMemCache **cachep );
```

Parameters

This function has the following parameters:

Table 18-76 ldap_memcache_init() function parameters

`ttl`	The maximum amount of time (in seconds) that an item can be cached. If `0`, there is no limit to the amount of time that an item can be cached.
`size`	Maximum amount of memory (in bytes) that the cache will consume. If `0`, the cache has no size limit.

Table 18-76 ldap_memcache_init() function parameters

baseDNs	An array of the base DN strings representing the base DNs of the search requests you want cached. • If not NULL, only the search requests with the specified base DNs will be cached. • If NULL, all search requests are cached.
thread_fns	An ldap_thread_fns structure specifying the functions that you want used to ensure that the cache is thread-safe. You should specify this if you have multiple threads that are using the same connection handle and cache. If you are not using multiple threads, pass NULL for this parameter.
cachep	When you call this function, the function sets this parameter to the pointer to the newly created LDAPMemCache structure.

Returns

One of the following values:

- LDAP_SUCCESS if the cache was successfully created.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_SIZELIMIT_EXCEEDED if the initial size of the cache (specified by the size argument) is too small.

Description

The ldap_memcache_init() function creates an in-memory, client-side cache that you can use to cache search requests.

The function passes back a pointer to an LDAPMemCache structure, which represents the cache. You should call the ldap_memcache_set() function to associate this cache with an LDAP connection handle (an LDAP structure).

On Windows NT, if the ldap_memcache_init() function returns an LDAP_PARAM_ERROR result code, verify that your client is using the version of the nsldap32v30.dll file provided with the Netscape LDAP SDK for C, v4.1. (Some of the Netscape servers, such as the Netscape Directory Server 3.0, install an older version of the nsldap32v30.dll file in the Windows system32 directory.) For example, you may want to copy the nsldap32v30.dll file provided with the Netscape LDAP SDK for C, v4.1 to your client's directory.

The cache uses search criteria as the key to cached items. When you send a search request, the cache checks the search criteria to determine if that request has been cached previously. If the request was cached, the search results are read from the cache.

To flush the cache, call the `ldap_memcache_flush()` function.

When you are done with the cache, you can free it from memory by calling the `ldap_memcache_destroy()` function.

See Also

`LDAPMemCache, ldap_memcache_set(), , ldap_memcache_update(), ldap_memcache_flush(), ldap_memcache_destroy().`

ldap_memcache_set()

Associates an in-memory cache with an LDAP connection handle.

Syntax

```
#include <ldap.h>
int ldap_memcache_set( LDAP *ld, LDAPMemCache *cache );
```

Parameters

This function has the following parameters:

Table 18-77 ldap_memcache_set() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`cache`	Pointer to an `LDAPMemCache` structure, which represents the cache that you want used for this connection.

Returns

One of the following values:

- `LDAP_SUCCESS` if the cache was successfully associated with the connection handle.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_SIZELIMIT_EXCEEDED` if the size of the cache is too small.

Description

The `ldap_memcache_set()` function associates a cache that you have created (by calling the `ldap_memcache_init()` function) with an LDAP connection handle.

After you call this function, search requests made over the specified LDAP connection will use this cache. Note that calling the `ldap_unbind()` function or the `ldap_unbind()` function will disassociate the cache from the LDAP connection handle.

You can call this function if you want to associate a cache with multiple LDAP connection handles. For example, you can call the `ldap_memcache_get()` function to get the cache associated with one connection, then you can call this function associate the cache with another connection.

See Also

`ldap_memcache_init(), ldap_memcache_get().`

ldap_memcache_update()

Checks the cache for items that have expired and removes them.

Syntax

```
#include <ldap.h>
void ldap_memcache_update( LDAPMemCache *cache );
```

Parameters

This function has the following parameters:

Table 18-78 ldap_memcache_update() function parameters

`cache`	Pointer to an `LDAPMemCache` structure, which represents the cache that you want to updated.

Description

The `ldap_memcache_update()` function checks the cache for items that have expired and removes them. This check is typically done as part of the way the cache normally works. You do not need to call this function unless you want to update the cache at this point in time.

This function is only useful in a multithreaded application, since it will not return until the cache is destroyed.

See Also

`ldap_memcache_flush()`.

ldap_memfree()

`ldap_memfree()` frees memory allocated by an LDAP API function call. For more information, see "Managing Memory."

Syntax

```
#include <ldap.h>
void ldap_memfree( void *p );
```

Parameters

This function has the following parameters:

Table 18-79 ldap_memfree() function parameter

p	Pointer to memory used by the LDAP library.

Example

The following section of code frees the memory allocated by the `ldap_get_dn()` function.

Code Example 18-29 ldap_memfree() code example

```
#include <ldap.h>
...
LDAP *ld;
char *dn;
LDAPMessage *entry;
...
/* Get the distinguished name (DN) for an entry */
dn = ldap_get_dn( ld, entry );
...
/* When you are finished working with the DN, free the memory allocated for the
DN */
ldap_memfree( dn );
...
```

See Also

`ldap_free_friendlymap()`, `ldap_free_urldesc()`, `ldap_msgfree()`, `ldap_value_free()`, `ldap_value_free_len()`.

ldap_modify()

Modifies an existing entry in the directory asynchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_modify_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_modify( LDAP *ld, const char *dn, LDAPMod **mods );
```

Parameters

This function has the following parameters:

Table 18-80 ldap_modify() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to modify.
`mods`	Pointer to a NULL-terminated array of pointers to `LDAPMod` structures representing the attributes that you want to modify.

Returns

Returns the message ID of the `ldap_modify()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`. For a list of possible result codes for an LDAP modify operation, see the result code documentation on the `ldap_modify_ext_s()` function.

Description

The `ldap_modify()` function updates an entry in the directory.

A newer version of this function, `ldap_modify_ext()`, is available in this release of the LDAP API. `ldap_modify()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_modify_ext()` instead of `ldap_modify()`.

If you want more information on `ldap_modify()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code uses the asynchronous `ldap_modify()` function to modify the entry for "Barbara Jensen" in the directory. The example makes the following changes to the entry:

1. Adds the homePhone attribute.
2. Changes the telephoneNumber attribute.
3. Removes the facsimileTelephoneNumber attribute.

Code Example 18-30 ldap_modify() code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the entry that you want to modify. */
char *dn = "uid=bjensen, ou=People, o=airius.com";

/* Values to add or change */
char *homePhone_values[] = { "555-1212", NULL };
char *telephoneNumber_values[] = { "869-5309", NULL };

...
/* Specify each change in separate LDAPMod structures */
attribute1.mod_type = "homePhone";
attribute1.mod_op = LDAP_MOD_ADD;
attribute1.mod_values = homePhone_values;
attribute2.mod_type = "telephoneNumber";
attribute2.mod_op = LDAP_MOD_REPLACE;
attribute2.mod_values = telephoneNumber_values;
attribute3.mod_type = "facsimileTelephoneNumber";
attribute3.mod_op = LDAP_MOD_DELETE;
attribute3.mod_values = NULL;
/* NOTE: When removing entire attributes, you need to specify a NULL value for
the mod_values field. */

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;
...
/* Set up the timeout period to wait for the "modify" operation */
tv.tv_sec = tv.tv_usec = 0;

/* Change the entry */
if ( ( msgid = ldap_modify( ld, dn, list_of_attrs ) ) == -1 ) {
  ldap_perror( ld, "ldap_modify" );
  return( 1 );
}

/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
```

Code Example 18-30 ldap_modify() code example

```
  ...
  /* do other work while waiting for the operation to complete */
  ...
}

/* Check the result to see if any errors occurred */
ldap_result2error( ld, result, 1 );
ldap_perror( ld, "ldap_modify" );
...
```

See Also

`ldap_modify_ext()`.

ldap_modify_ext()

Modifies an existing entry in the directory asynchronously.

Syntax

```
#include <ldap.h>
int ldap_modify_ext( LDAP *ld, const char *dn, LDAPMod **mods,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-81 ldap_modify_ext() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to modify.
`mods`	Pointer to a `NULL`-terminated array of pointers to `LDAPMod` structures representing the attributes that you want to modify.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.

Table 18-81 ldap_modify_ext() function parameters

`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`msgidp`	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call `ldap_result()` and `ldap_parse_result()` functions.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The `ldap_modify_ext()` modifies an entry in the directory asynchronously.

This function is a new version of the `ldap_modify()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_modify()`.

To make changes to an entry to the directory, you need to specify the following information:

- A unique DN identifying the new entry.

  Use the `dn` argument to specify the DN of the entry that you want to modify.

- A set of attributes for the new entry.

Create an `LDAPMod` structure for change that you want to make to an attribute. Create an array of these `LDAPMod` structures and pass the array as the `mods` argument.

For information on using the `LDAPMod` structure to specify a change, see `LDAPMod`.

The function `ldap_modify_ext()` is asynchronous; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_modify_ext_s()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP modify operation, you need to call the `ldap_result()` function and the `ldap_parse_result()` function. (See "Calling Asynchronous Functions" for details.) For a list of possible result codes for an LDAP modify operation, see the result code documentation for the `ldap_modify_ext_s()` function.

For more information on modifying entries in the directory, see "Modifying an Entry."

Example

See the example under "Example: Modifying an Entry in the Directory (Asynchronous)."

See Also

`ldap_modify_ext_s(), ldap_result(), ldap_parse_result(), LDAPMod.`

ldap_modify_ext_s()

Modifies an existing entry in the directory synchronously.

Syntax

```
#include <ldap.h>
int ldap_modify_ext_s( LDAP *ld, const char *dn, LDAPMod **mods,
  LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-82 ldap_modify_ext_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.

Table 18-82 ldap_modify_ext_s() function parameters

`dn`	Distinguished name (DN) of the entry to modify.
`mods`	Pointer to a NULL-terminated array of pointers to `LDAPMod` structures for the attributes you want modified.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client.”)

The following result codes can be returned by the Directory Server when processing an LDAP modify request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- `LDAP_OPERATIONS_ERROR` may be sent by the Directory Server for general errors encountered by the server when processing the request.

- LDAP_PROTOCOL_ERROR if the modify request did not comply with the LDAP protocol. The Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
  - The server encountered an error when decoding your client's BER-encoded request.
  - An unknown modification operator (other that LDAP_MOD_ADD, LDAP_MOD_DELETE, and LDAP_MOD_REPLACE) is specified in the request.
  - The modification operator is LDAP_MOD_ADD and no values are specified.
  - No modifications are specified.
- LDAP_CONSTRAINT_VIOLATION may be sent by the Directory Server if your client is attempting to modify the userpassword attribute in the following situations:
  - The server is configured to require a minimum password length and the new value of the userpassword attribute is shorter than the minimum length.
  - The server is configured to keep track of previous passwords, and the new value of the userpassword attribute is the same as a previous value of that attribute.
  - The new value of the userpassword attribute is the same as the value of the uid, cn, sn, givenname, ou, or mail attributes. (Using a password that is the same as your user id or email address would make the password trivial and easy to crack.)
- LDAP_UNWILLING_TO_PERFORM may be sent by the Directory Server in the following situations:
  - Your client is attempting to modify the aci attribute and an error occurs.
  - The value of the userpassword attribute needs to be changed. (The password has expired.)
  - Your client is attempting to add attributes to the configuration DSE (DSA-specific entry, where DSA stands for Directory-Specific Agent).
  - Your client is attempting to replace attributes in the schema DSE.
  - The server's database is read-only.

- LDAP_REFERRAL may be sent by the Directory Server if the DN specified by the dn argument identifies an entry not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is uid=bjensen, ou=European Sales, o=airius.com, all entries under ou=European Sales might be handled by a different directory server.)
- LDAP_NO_SUCH_ATTRIBUTE may be sent by the Directory Server if the attribute that you want to modify (add, replace, or delete) does not exist.
- LDAP_INVALID_SYNTAX may be sent by the Directory Server if your client is modifying the schema DSE and no object class or attribute type is specified.
- LDAP_NO_SUCH_OBJECT may be sent by the Directory Server if the entry that you want modified does not exist.
- LDAP_INSUFFICIENT_ACCESS may be sent by the Directory Server if the DN that your client is authenticated as does not have the access rights to modify the entry.
- LDAP_TYPE_OR_VALUE_EXISTS may be sent by the Directory Server if your client is attempting to add an attribute to an entry and the attribute values already exist in the entry.
- LDAP_OBJECT_CLASS_VIOLATION may be sent by the Directory Server if the modified entry does not comply with the Directory Server schema (for example, if one or more required attributes are not specified).
- LDAP_NOT_ALLOWED_ON_RDN may be sent by the Directory Server if the modified entry no longer contains attributes for each DN component.

Note that the Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The ldap_modify_ext_s() modifies an entry in the directory synchronously.

This function is a new version of the ldap_modify_s() function. If you are writing a new LDAP client, you should call this function instead of ldap_modify_s().

To make changes to an entry to the directory, you need to specify the following information:

- A unique DN identifying the new entry.

  Use the dn argument to specify the DN of the entry that you want to modify.

- A set of attributes for the new entry.

Create an LDAPMod structure for change that you want to make to an attribute. Create an array of these LDAPMod structures and pass the array as the mods argument.

For information on using the LDAPMod structure to specify a change, see LDAPMod.

ldap_modify_ext_s() is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function ldap_modify_ext() instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For more information on modifying entries in the directory, see "Modifying an Entry."

Example

See the example under "Example: Modifying an Entry in the Directory (Synchronous)."

See Also

ldap_modify_ext(), LDAPMod.

ldap_modify_s()

Modifies an existing entry in the directory synchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use ldap_modify_ext_s() instead.

Syntax

```
#include <ldap.h>
int ldap_modify_s( LDAP *ld, const char *dn, LDAPMod **mods );
```

Parameters

This function has the following parameters:

Table 18-83 ldap_modify_s() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	Distinguished name (DN) of the entry to modify.

Table 18-83 ldap_modify_s() function parameters

mods	Pointer to a NULL-terminated array of pointers to LDAPMod structures for the attributes you want modified.

Returns

For a list of possible result codes for an LDAP modify operation, see the result code documentation for the ldap_modify_ext_s() function.

Description

The ldap_modify_s() function updates an entry in the directory.

A newer version of this function, ldap_modify_ext_s(), is available in this release of the LDAP API. ldap_modify_s() (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use ldap_modify_ext_s() instead of ldap_modify_s().

If you want more information on ldap_modify_s(), refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code uses the synchronous ldap_modify_s() function to makes the following changes to the "Barbara Jensen" entry:

1. Adds the homePhone attribute.
2. Changes the telephoneNumber attribute.
3. Removes the facsimileTelephoneNumber attribute.

Code Example 18-31 ldap_modify_s() code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;
LDAPControl **srvrctrls, **clntctrls;

/* Distinguished name of the entry that you want to modify. */
char *dn = "uid=bjensen, ou=People, o=airius.com";

/* Values to add or change */
char *homePhone_values[] = { "555-1212", NULL };
char *telephoneNumber_values[] = { "869-5309", NULL };

...
/* Specify each change in separate LDAPMod structures */
```

Code Example 18-31 ldap_modify_s() code example

```
attribute1.mod_type = "homePhone";
attribute1.mod_op = LDAP_MOD_ADD;
attribute1.mod_values = homePhone_values;
attribute2.mod_type = "telephoneNumber";
attribute2.mod_op = LDAP_MOD_REPLACE;
attribute2.mod_values = telephoneNumber_values;
attribute3.mod_type = "facsimileTelephoneNumber";
attribute3.mod_op = LDAP_MOD_DELETE;
attribute3.mod_values = NULL;
/* NOTE: When removing entire attributes, you need to specify a NULL value for
the mod_values or mod_bvalues field. */

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;
...
/* Change the entry */
if ( ldap_modify_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_modify_s" );
  return( 1 );
}
...
```

See Also

`ldap_modify_ext_s()`.

ldap_modrdn()

The `ldap_modrdn()` function is a deprecated function. Use the `ldap_rename()` or `ldap_modrdn2()` functions instead.

ldap_modrdn_s()

The `ldap_modrdn_s()` function is a deprecated function. Use the `ldap_rename_s()` or `ldap_modrdn2_s()` functions instead.

ldap_modrdn2()

Changes the relative distinguished name (RDN) of an entry in the directory asynchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_rename()` instead.

Syntax

```
#include <ldap.h>
int ldap_modrdn2( LDAP *ld, const char *dn, const char *newrdn,
  int deleteoldrdn );
```

Parameters

This function has the following parameters:

Table 18-84 ldap_modrdn2() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to modify.
`newrdn`	New relative distinguished name (RDN) to assign to the entry.
`deleteoldrdn`	If this is a non-zero value, the old RDN is not retained as a value in the modified entry. If `0`, the old RDN is retained as an attribute in the modified entry.

Returns

Returns the message ID of the `ldap_modrdn2()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`. For a list of possible result codes, see the result code documentation for the `ldap_rename_s()` function.

Description

The `ldap_modrdn2()` function modifies the relative distinguished name (RDN) of an entry in a directory and lets you specify whether or not the old RDN is retained as an attribute of the entry.

A newer version of this function, `ldap_rename()`, is available in this release of the LDAP API. `ldap_modrdn2()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_rename()` instead of `ldap_modrdn2()`.

If you want more information on `ldap_modrdn2()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code uses the asynchronous `ldap_modrdn2()` function to change the RDN of an entry from “`uid=bjensen`” to "`uid=babs`". The code removes the existing RDN "`bjensen`" from the `uid` attribute of the entry.

Code Example 18-32 ldap_modrdn2() code example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the entry that you want to rename. */
char *dn = "uid=bjensen, ou=People, o=airius.com";

/* New relative distinguished name (RDN) of the entry */
char *rdn = "uid=babs";
...
/* Set up the timeout period to wait for the "modify RDN" operation */
tv.tv_sec = tv.tv_usec = 0;

/* Rename the entry */
if ( ( msgid = ldap_modrdn2( ld, dn, rdn, 1 ) ) == -1 ) {
  ldap_perror( ld, "ldap_modrdn2" );
  return( 1 );
}

/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
  ...
  /* do other work while waiting for the operation to complete */
  ...
}

/* Check the result to see if any errors occurred */
ldap_result2error( ld, result, 1 );
ldap_perror( ld, "ldap_modrdn2" );
...
```

See Also

`ldap_rename()`.

ldap_modrdn2_s()

Changes the relative distinguished name (RDN) of an entry in the directory synchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_rename_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_modrdn2_s( LDAP *ld, const char *dn,
  const char *newrdn, int deleteoldrdn );
```

Parameters

This function has the following parameters:

Table 18-85 ldap_modrdn2_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to modify.
`newrdn`	New relative distinguished name (RDN) to assign to the entry.
`deleteoldrdn`	If this is a non-zero value, the old RDN is not retained as a value in the modified entry. If `0`, the old RDN is retained as an attribute in the modified entry.

Returns

For a list of possible result codes, see the result code documentation for the `ldap_rename_s()` function.

Description

The `ldap_modrdn2_s()` function modifies the relative distinguished name (RDN) of an entry in a directory and lets you specify whether or not the old RDN is retained as an attribute of the entry.

A newer version of this function, `ldap_rename_s()`, is available in this release of the LDAP API. `ldap_modrdn2_s()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_rename_s()` instead of `ldap_modrdn2_s()`.

If you want more information on `ldap_modrdn2_s()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code uses the synchronous `ldap_modrdn2_s()` function to change the RDN of an entry from "`uid=bjensen`" to "`uid=babs`". The code removes the existing RDN "`babs`" from the `uid` attribute of the entry.

Code Example 18-33 ldap_modrdn2_s() code example

```
#include <ldap.h>
...
LDAP *ld;

/* Distinguished name of the entry that you want to rename. */
char *dn = "uid=bjensen, ou=People, o=airius.com";

/* New relative distinguished name (RDN) of the entry */
char *rdn = "uid=babs";
...
/* Rename the entry */
if ( ldap_modrdn2_s( ld, dn, rdn, 1 ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_modrdn2_s" );
  return( 1 );
}
...
```

See Also

`ldap_rename_s()`.

ldap_mods_free()

The `ldap_mods_free()` function frees the `LDAPMod` structures that you've allocated to add or modify entries. You need to call this function only if you've allocated memory for these structures yourself.

Syntax

```
#include <ldap.h>
void ldap_mods_free( LDAPMod **mods, int freemods );
```

Parameters

This function has the following parameters:

Table 18-86 ldap_mods_free() function parameters

`mods`	Pointer to a NULL-terminated array of pointers to `LDAPMod` structures.
`freemods`	If this is a non-zero value, frees the array of pointers as well as the `LDAPMod` structures. If 0, just frees the `LDAPMod` structures.

Example

The following example allocates memory for LDAPMod structures and frees them when done.

Code Example 18-34 ldap_mods_free() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *dn;
int i, msgid;
LDAPMod **mods;
...
/* Construct the array of values to add */
mods = ( LDAPMod ** ) malloc(( NMODS + 1 ) * sizeof( LDAPMod * ));
if ( mods == NULL ) {
  fprintf( stderr, "Cannot allocate memory for mods array\n" );
}
for ( i = 0; i < NMODS; i++ ) {
  if (( mods[ i ] = ( LDAPMod * ) malloc( sizeof( LDAPMod ))) == NULL) {
    fprintf( stderr, "Cannot allocate memory for mods element\n" );
    exit( 1 );
  }
}
...
/* Code for filling the structures goes here. */
...
/* Initiate the add operation */
if (( msgid = ldap_add( ld, dn, mods )) < 0 ) {
  ldap_perror( ld, "ldap_add" );
  ldap_mods_free( mods, 1 );
  return( 1 );
}
...
```

ldap_msgfree()

The ldap_msgfree() function frees the memory allocated for a result by ldap_result() or ldap_search_s(). For more information, see “Freeing the Results of a Search.”

Syntax

```
#include <ldap.h>
int ldap_msgfree( LDAPMessage *lm );
```

Parameters

This function has the following parameters:

Table 18-87 ldap_msgfree() function parameters

`lm`	ID of the result to be freed from memory.

Returns

One of the following values:

- If successful, returns the type of result freed. Possible types of results are listed below.

Table 18-88 ldap_msgfree() return codes

`LDAP_RES_BIND`	Results from an LDAP bind operation.
`LDAP_RES_SEARCH_ENTRY`	Entry returned by an LDAP search operation.
`LDAP_RES_SEARCH_RESULT`	Results from an LDAP search operation.
`LDAP_RES_MODIFY`	Results from an LDAP modify operation.
`LDAP_RES_ADD`	Results from an LDAP add operation.
`LDAP_RES_DELETE`	Results from an LDAP delete operation.
`LDAP_RES_MODRDN`, `LDAP_RES_RENAME`	Results from an LDAP modify DN operation.
`LDAP_RES_COMPARE`	Results from an LDAP compare operation.
`LDAP_RES_SEARCH_REFERENCE`	Search reference returned by an LDAP search operation.
`LDAP_RES_EXTENDED`	Results from an LDAP extended operation.
`LDAP_RES_ANY`	Results from any asynchronous LDAP operation.

- If unsuccessful, returns the LDAP error code for the operation. (See Chapter 19, “Result Codes” for a complete listing.)
- If the operation times out, returns `LDAP_SUCCESS`.

Example

The following example frees the results of a search.

Code Example 18-35 ldap_msgfree() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
char *get_attr[] = { "cn", "mail", NULL };
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
get_attr, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}
...
/* Free the results when done */
ldap_msgfree( result );
...
```

See Also

`ldap_result()`, `ldap_search_s()`.

ldap_msgid()

The `ldap_msgid()` function determines the message ID of a result obtained by calling `ldap_result()` or `ldap_search_s()`.

Syntax

```
#include <ldap.h>
int ldap_msgid( LDAPMessage *lm );
```

Parameters

This function has the following parameters:

Table 18-89 ldap_msgid() function parameters

`lm`	ID of the result to check.

Returns

One of the following values:

- The message ID if successful.
- `-1` if unsuccessful.

Example

The following example prints the message ID from the result obtained from a synchronous LDAP search operation.

Code Example 18-36 ldap_msgid() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
...
/* Perform a search */
if ( ldap_search_s( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE, MY_FILTER,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get and print the message ID */
if ( ldap_msgid( result ) != -1 ) {
  printf( "Message ID: %d\n" );
} else {
  printf( "An error occurred.\n" );
}
...
```

See Also

`ldap_msgtype(), ldap_result(), ldap_search_s().`

ldap_msgtype()

The `ldap_msgtype()` function determines the type of result obtained by calling `ldap_result()` or `ldap_search_s()`.

Syntax

```
#include <ldap.h>
int ldap_msgtype( LDAPMessage *lm );
```

Parameters

This function has the following parameters:

Table 18-90 ldap_msgtype() function parameters

`lm`	ID of the result to check.

Returns

One of the following values:

- `LDAP_RES_BIND` indicates that the `LDAPMessage` structure contains the result of an LDAP bind operation.
- `LDAP_RES_SEARCH_ENTRY` indicates that the `LDAPMessage` structure contains an entry found during an LDAP search operation.
- `LDAP_RES_SEARCH_REFERENCE` indicates that the `LDAPMessage` structure contains an LDAPv3 search reference (a referral to another LDAP server) found during an LDAP search operation.
- `LDAP_RES_SEARCH_RESULT` indicates that the `LDAPMessage` structure contains the result of an LDAP search operation.
- `LDAP_RES_MODIFY` ndicates that the `LDAPMessage` structure contains the result of an LDAP modify operation.
- `LDAP_RES_ADD` indicates that the `LDAPMessage` structure contains the result of an LDAP add operation.
- `LDAP_RES_DELETE` indicates that the `LDAPMessage` structure contains the result of an LDAP delete operation.
- `LDAP_RES_MODRDN` or `LDAP_RES_RENAME` indicates that the `LDAPMessage` structure contains the result of an LDAP modify DN operation.
- `LDAP_RES_COMPARE` indicates that the `LDAPMessage` structure contains the result of an LDAP compare operation.
- `LDAP_RES_EXTENDED` indicates that the `LDAPMessage` structure contains the result of an LDAPv3 extended operation.
- -1 indicates that the `lm` argument is not a pointer to a valid `LDAPMessage` structure.

Example

The following example prints the message type for a result obtained from a synchronous LDAP search operation.

Code Example 18-37 ldap_msgtype() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
int msgtype;
...
/* Perform a search */
if ( ldap_search_s( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE, MY_FILTER,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Get and print the message type */
msgtype = ldap_msgtype( result );
if ( msgtype != -1 ) {
  printf( "Message type: %d\n", msgtype );
} else {
  printf( "An error occurred.\n" );
}
...
```

See Also

`ldap_msgid()`, `ldap_result()`, `ldap_search_s()`.

ldap_multisort_entries()

The `ldap_multisort_entries()` function sorts a chain of entries retrieved from an LDAP search call (`ldap_search_s()` or `ldap_result()`) by a specified set of attributes in the entries (or by DN if you don't specify a set of attributes).

For additional information, see "Sorting the Search Results."

Syntax

```
#include <ldap.h>
int ldap_multisort_entries( LDAP *ld, LDAPMessage **chain,
  char **attr, LDAP_CMP_CALLBACK *cmp );
```

Parameters

This function has the following parameters:

Table 18-91 ldap_multisort_entries() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`chain`	Chain of entries returned by the `ldap_result()` or `ldap_search_s()` function.
`attr`	Array of attributes to use when sorting the results. If `NULL`, results are sorted by distinguished name (DN).
`cmp`	Comparison function used when sorting the values.

Returns

One of the following values:

- `0` if successful.
- `-1` if memory cannot be allocated by this function. (The error code `LDAP_NO_MEMORY` is set in the `LDAP` structure. To get the error code, call the `ldap_get_lderrno()` function.)
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function

Example

The following section of code sorts entries first by the `roomNumber` attribute, then by the `telephoneNumber` attribute.

Code Example 18-38 ldap_multisort_entries() code example

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
char *attrs[3];
attrs[0] = "roomNumber";
attrs[1] = "telephoneNumber";
attrs[2] = NULL;
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter, NULL, 0,
&result ) != LDAP_SUCCESS ) {
```

Code Example 18-38 ldap_multisort_entries() code example

```
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}

/* Sort the results, using strcasecmp */
if ( ldap_multisort_entries( ld, &result, attrs, strcasecmp ) != LDAP_SUCCESS
) {
  ldap_perror( ld, "ldap_multisort_entries" );
  return( 1 );
}
...
```

See Also

`ldap_result()`, `ldap_search_s()`, `ldap_sort_entries()`.

ldap_next_attribute()

The `ldap_next_attribute()` function returns the name of the next attribute in a entry returned by `ldap_first_entry()` or `ldap_next_entry()`.

The `ldap_first_attribute()` function returns a pointer to a `BerElement`. You use this pointer with `ldap_next_attribute()` to iterate through the list of elements. After the last call to `ldap_next_element()`, you should free the `BerElement` pointer using `ldap_ber_free()`. When calling `ldap_ber_free()`, make sure to specify that the buffer is not freed (pass `0` for the `freebuf` parameter).

For more information, see “Getting Attributes from an Entry.”

Syntax

```
#include <ldap.h>
char * ldap_next_attribute( LDAP *ld, LDAPMessage *entry,
  BerElement *ber);
```

Parameters

This function has the following parameters:

Table 18-92 ldap_next_attribute() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`entry`	Pointer to the `LDAPMessage` structure representing the entry returned by the `ldap_first_entry()` or `ldap_next_entry()` function.

Table 18-92 ldap_next_attribute() function parameters

ber	A pointer to a BerElement allocated to keep track of its current position. Pass this pointer to subsequent calls to ldap_next_attribute() to step through the entry's attributes.

Returns

One of the following values:

- If successful, returns the name of the next attribute in an entry. When you are done using this data, you should free the memory by calling the ldap_memfree() function.
- If no more attributes exist in the entry, returns a NULL.
- If unsuccessful, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the ldap_get_lderrno() function. (See Chapter 19, "Result Codes" for a complete listing of error codes.)

Example

See the example under ldap_first_attribute().

See Also

ldap_first_attribute(), ldap_getfirstfilter(), ldap_next_entry().

ldap_next_entry()

Returns a pointer to the LDAPMessage structure representing the next directory entry in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_next_entry( LDAP *ld, LDAPMessage *entry );
```

Parameters

This function has the following parameters:

Table 18-93 ldap_next_entry() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Pointer to an LDAPMessage structure in a chain of search results.

Returns

One of the following values:

- If successful, returns the pointer to the next LDAPMessage structure of the type LDAP_RES_SEARCH_ENTRY in a chain of search results.
- If no more LDAPMessage structures of the type LDAP_RES_SEARCH_ENTRY are in the chain or if the function is unsuccessful, returns a NULLMSG.

Description

The ldap_next_entry() function returns a pointer to the LDAPMessage structure representing the next directory entry in a chain of search results. Messages containing directory entries have the type LDAP_RES_SEARCH_ENTRY.

You can use this function in conjunction with the ldap_first_entry() function to iterate through the directory entries in a chain of search results. These functions skip over any messages in the chain that do not have the type LDAP_RES_SEARCH_ENTRY.

For more information, see “Iterating Through a Chain of Results.”

See Also

ldap_first_entry().

ldap_next_message()

Returns a pointer to the next LDAPMessage structure in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_next_message( LDAP *ld, LDAPMessage *msg );
```

Parameters

This function has the following parameters:

Table 18-94 ldap_next_message() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
msg	Pointer to an LDAPMessage structure in a chain of search results.

Returns

One of the following values:

- If successful, returns the pointer to the next LDAPMessage structure in a chain of search results.
- If no more LDAPMessage structures are in the chain or if the function is unsuccessful, returns a NULLMSG.

Description

The ldap_next_message() function returns a pointer to the next LDAPMessage structure in a chain of search results.

You can use this function in conjunction with the ldap_first_message() function to iterate through the chain of search results. You can call the ldap_msgtype() function to determine if each message contains a matching entry (a message of the type LDAP_RES_SEARCH_ENTRY) or a search reference (a message of the type LDAP_RES_SEARCH_REFERENCE).

For more information, see "Iterating Through a Chain of Results."

Example

See the examples under "Example: Searching the Directory (Synchronous)" and "Example: Searching the Directory (Asynchronous)."

See Also

ldap_first_message().

ldap_next_reference()

Returns a pointer to the LDAPMessage structure representing the next search reference in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_next_reference( LDAP *ld, LDAPMessage *ref );
```

Parameters

This function has the following parameters:

Table 18-95 ldap_next_reference() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
msg	Pointer to an LDAPMessage structure in a chain of search results.

Returns

One of the following values:

- If successful, returns the pointer to the next `LDAPMessage` structure of the type `LDAP_RES_SEARCH_REFERENCE` in a chain of search results.
- If no more `LDAPMessage` structures of the type `LDAP_RES_SEARCH_REFERENCE` are in the chain or if the function is unsuccessful, returns a `NULLMSG`.

Description

The `ldap_next_reference()` function returns a pointer to the `LDAPMessage` structure representing the next search reference in a chain of search results. Messages containing search references have the type `LDAP_RES_SEARCH_REFERENCE`.

You can use this function in conjunction with the `ldap_first_reference()` function to iterate through the search references in a chain of search results. These functions skip over any messages in the chain that do not have the type `LDAP_RES_SEARCH_REFERENCE`.

For more information, see “Iterating Through a Chain of Results.”

See Also

`ldap_first_reference()`.

ldap_parse_entrychange_control()

Examines a list of controls returned from a persistent search operation, retrieves an entry change control, and parses that control for information (such as the type of change made to the entry and the change number).

This function implements an extension to the LDAPv3 protocol. Entry change notification is an optional LDAP server feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension.

Syntax

```
#include <ldap.h>
int ldap_parse_entrychange_control( LDAP *ld,
  LDAPControl **ctrls, int *chgtypep, char **prevdnp,
  int *chgnumpresentp, long *chgnump );
```

Parameters

This function has the following parameters:

Table 18-96 ldap_parse_entrychange_control() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`ctrlp`	An array of controls returned by the server. You obtain these controls by calling the `ldap_get_entry_controls()` function on an entry returned by the server.
`changetypes`	Pointer to an integer specifying the type of change made to the entry. This field can have one of the following values: • `LDAP_CHANGETYPE_ADD` specifies that the entry was added to the directory. • `LDAP_CHANGETYPE_DELETE` specifies that the entry was deleted from the directory. • `LDAP_CHANGETYPE_MODIFY` specifies that the entry was modified. • `LDAP_CHANGETYPE_MODDN` specifies that the DN or RDN of the entry was changed (a modify RDN or modify DN operation was performed).
`prevdnp`	Pointer to the previous DN of the entry, if the `changetypes` argument is `LDAP_CHANGETYPE_MODDN`. (If the `changetypes` argument has a different value, this argument is set to `NULL`.) When done, you can free this by calling the `ldap_memfree()` function.
`chgnumpresentp`	Pointer to an integer specifying whether or not to the change number is included in the control. The parameter can have the following possible values: • `0` specifies that the change number is not included. • A non-zero value specifies that the change number is included and is available as the `chgnump` argument.
`chgnump`	Pointer to the change number identifying the change made to the entry, if `chgnumpresentp` points to a non-zero value.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.

- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_DECODING_ERROR if an error occurred when BER-decoding the control.

Description

Call the ldap_parse_entrychange_control() function to parse an entry returned from a persistent search operation and retrieve an entry change control.

Call this function after receiving an entry from a persistent search and retrieving the controls from the entry (call ldap_get_entry_controls() to get the controls).

See Also

ldap_create_persistentsearch_control(), ldap_get_entry_controls().

ldap_parse_extended_result()

Parses the results of an LDAP extended operation and retrieves the OID and data returned by the server.

This function implements an extension to the LDAPv3 protocol. Extended operations might not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension. See "Determining the Extended Operations Supported" to determine if the server supports extended operations.

Syntax

```
#include <ldap.h>
int ldap_parse_extended_result( LDAP *ld, LDAPMessage *res,
  char **retoidp, struct berval **retdatap, int freeit );
```

Parameters

This function has the following parameters:

Table 18-97 ldap_parse_extended_result() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
res	Pointer to the LDAPMessage structure containing the results of an LDAP operation.
retoidp	Pointer to the object identifier (OID) returned by the server after performing the extended operation. When done, you can free this by calling the ldap_memfree() function.

Table 18-97 ldap_parse_extended_result() function parameters

`retdatap`	Pointer to the pointer to a berval structure containing the data returned by the server after performing the extended operation. When done, you can free this by calling the `ber_bvfree()` function.
`freeit`	Specifies whether or not to free the results of the operation (the LDAPMessage structure specified by the `res` argument). The parameter can have the following possible values: • `0` specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values, which indicates the result of parsing the server's response (this value does not apply to the LDAP extended operation itself):

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.
- `LDAP_NOT_SUPPORTED` if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

After you call the `ldap_extended_operation()` function and the `ldap_result()` function, you can pass the results to the `ldap_parse_extended_result()` function. `ldap_parse_extended_result()` parses the server's response to an extended operation.

This function gets the following data from the server's response:

- The extended operation OID received from the server is passed back as the `retoidp` argument.
- The data received from the server is passed back in the berval structure as the `retdatap` argument.
- The LDAP result code for the LDAP extended operation is placed in the `ld` structure. You can get the result code by calling the `ldap_get_lderrno()` function.

For a list of possible result codes for an LDAP extended operation, see the result code documentation for the `ldap_extended_operation_s()` function.

The LDAP server must support the extended operation. The Netscape Directory Server 3.0 supports a server plug-in interface that you can use to add support for extended operations to the server. For details, see the *Netscape Directory Server 3.0 Programmer's Guide*.

See Also

`ldap_extended_operation(), ldap_get_lderrno().`

ldap_parse_reference()

Parses search references from the results received from an LDAP server.

Search references are part of the LDAPv3 protocol. When calling this function, make sure that you are working with a server that supports the LDAPv3 protocol.

Syntax

```
#include <ldap.h>
int ldap_parse_reference( LDAP *ld, LDAPMessage *ref,
  char ***referralsp, LDAPControl ***serverctrlsp,
  int freeit );
```

Parameters

This function has the following parameters:

Table 18-98 ldap_parse_reference() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`ref`	Pointer to an `LDAPMessage` structure of the type `LDAP_RES_SEARCH_REFERENCE`.
`referralsp`	Pointer to an array of strings representing the referrals found by an LDAP search operation and returned by the server (applicable only if the LDAP operation was a search operation). When done, you can free this by calling the `ldap_value_free()` function.
`serverctrlsp`	Pointer to an array of `LDAPControl` structures, which represent the LDAPv3 server controls returned by the server. When done, you can free this by calling the `ldap_controls_free()` function.

Table 18-98 ldap_parse_reference() function parameters

`freeit`	Specifies whether or not to free the results of the operation (the `LDAPMessage` structure specified by the `res` argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values:

- `LDAP_SUCCESS` if the referral URLs were retrieved successfully.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded message.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NO_MEMORY` if memory cannot be allocated.

Description

The `ldap_parse_reference()` function parses the referral URLs from an `LDAPMessage` structure of the type `LDAP_RES_SEARCH_REFERENCE`.

ldap_parse_result()

Parses the results of an LDAP operation received from an LDAP server.

Syntax

```
#include <ldap.h>
int ldap_parse_result( LDAP *ld, LDAPMessage *res,
  int *errcodep, char **matcheddnp, char **errmsgp,
  char ***referralsp, LDAPControl ***serverctrlsp, int freeit);
```

Parameters

This function has the following parameters:

Table 18-99 ldap_parse_result() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`res`	Pointer to the `LDAPMessage` structure containing the results of an LDAP operation.

Table 18-99 ldap_parse_result() function parameters

`errcodep`	Pointer to the LDAP result code specifying the result of the operation.
`matcheddnp`	Pointer to a string specifying the portion of a DN that finds an existing entry (in cases where the server cannot find the entry specified by a DN). See “Receiving the Portion of the DN Matching an Entry” for details. When done, you can free this by calling the `ldap_memfree()` function.
`errmsgp`	Pointer to an additional error message string sent from the server. When done, you can free this by calling the `ldap_memfree()` function.
`referralsp`	Pointer to an array of strings representing the referrals returned by the server. When done, you can free this by calling the `ldap_value_free()` function.
`serverctrlsp`	Pointer to an array of `LDAPControl` structures, which represent the LDAPv3 server controls returned by the server. When done, you can free this by calling the `ldap_controls_free()` function.
`freeit`	Specifies whether or not to automatically free the results of the operation (the `LDAPMessage` structure specified by the `res` argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values:

- `LDAP_SUCCESS` if the results were parsed successfully.
- `LDAP_NO_RESULTS_RETURNED` if the specified `LDAPMessage` structure does not contain the result of an LDAP operation (for example, if it contains an entry, search reference, or chain of search results instead of the result of an LDAP operation).
- `LDAP_MORE_RESULTS_TO_RETURN` if the result in the `LDAPMessage` structure is part of a chain of results and the last result is not included.

- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded message.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The ldap_parse_result() function parses the results of an LDAP operation (received from an LDAP server) and retrieves the following information:

- The LDAP result code that indicates the result of the LDAP operation (errcodep).
- An additional error message (optional) sent by the server (errmsgp).
- The portion of the DN that finds an entry, if the server is unable to find an entry from a DN that you specify (matcheddnp) (see "Receiving the Portion of the DN Matching an Entry" for details).
- A set of referrals, if the server does not contain the entries that you've specified and if the server is configured to refer clients to other servers (referralsp).
- a set of server response controls that are relevant to the operation (serverctrlsp).

Calling this function creates an array of LDAPControl structures that you can pass to subsequent API functions (such as the ldap_parse_sort_control() function).

Note that this function is not intended to be used to parse entries and search references. (Use the ldap_msgtype() function to determine the type of result contained in the LDAPMessage structure.)

- If the result is an entry returned as a search result, call the ldap_first_entry() function to retrieve the entry.
- If the result is a search reference, call the ldap_parse_reference() function to retrieve the reference.

Example

See the examples in the following sections:

- For an example of parsing the results of an asynchronous LDAP add operation, see "Example: Adding an Entry to the Directory (Asynchronous)."
- For an example of parsing the results of an asynchronous LDAP modify operation, see "Example: Modifying an Entry in the Directory (Asynchronous)."

- For an example of parsing the results of an asynchronous LDAP delete operation, see “Example: Deleting an Entry from the Directory (Asynchronous).”
- For an example of parsing the results of an asynchronous LDAP modify DN operation, see “Example: Renaming an Entry in the Directory (Asynchronous).”
- For an example of parsing the results of an asynchronous LDAP search operation, see “Example: Searching the Directory (Asynchronous).”
- For an example of parsing the results of an asynchronous LDAP bind operation, see “Performing an Asynchronous Authentication Operation.”

See Also

`ldap_result()`.

ldap_parse_sasl_bind_result()

Parses the results of an LDAP SASL bind operation and retrieves data (such as a challenge) returned by the server. This function also gets the LDAP result code for the SASL bind operation and sets it in the `ld` structure. (You can retrieve it by calling the `ldap_get_lderrno()` function.)

SASL authentication is part of the LDAPv3 protocol. When calling this function, make sure that you are working with a server that supports the LDAPv3 protocol.

Syntax

```
#include <ldap.h>
int ldap_parse_sasl_bind_result( LDAP *ld, LDAPMessage *res,
  struct berval **servercredp, int freeit );
```

Parameters

This function has the following parameters:

Table 18-100 ldap_parse_sals_bind_result() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`res`	Pointer to the `LDAPMessage` structure containing the results of an LDAP operation.
`servercredp`	Pointer to a pointer to an `berval` structure containing any challenge or credentials returned by the server. When done, you can free this by calling the `ber_bvfree()` function.

Table 18-100 ldap_parse_sals_bind_result() function parameters

`freeit`	Specifies whether or not to free the results of the operation (the `LDAPMessage` structure specified by the `res` argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values, which indicates the result of parsing the server's response (this value does not apply to the SASL bind operation itself):

- `LDAP_SUCCESS` if the results were parsed successfully.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NOT_SUPPORTED` if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded message.

Description

After you call the `ldap_sasl_bind()` function and the `ldap_result()` function, you can pass the results to the `ldap_parse_sasl_bind_result()` function parse the results from the server.

This function gets the following data from the server's response:

- The challenge or credentials sent back from the server are passed back in the berval structure as the `servercredp` argument.
- The LDAP result code for the SASL bind operation is placed in the `ld` structure. You can get the result code by calling the `ldap_get_lderrno()` function.

  For a list of possible result codes for an LDAP SASL bind operation, see the result code documentation for the `ldap_sasl_bind_s()` function.

If the result code is `LDAP_SASL_BIND_IN_PROGRESS`, you can call `ldap_sasl_bind()` again to send a response to the server's challenge and call `ldap_result()` and `ldap_parse_sasl_bind_result()` again to get the next challenge from the server.

The LDAP server must support authentication through SASL mechanisms. The Directory Server supports a server plug-in interface that you can use to add SASL support to the server. For details, see the *iPlanet Directory Server Programmer's Guide*.

See Also

`ldap_sasl_bind(), ldap_get_lderrno().`

ldap_parse_sort_control()

Parses the result returned from a search operation that used a server control for sorting search results.

This function implements an extension to the LDAPv3 protocol. Server-side sorting is an optional LDAP server feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension.

Syntax

```
#include <ldap.h>
int ldap_parse_sort_control( LDAP *ld, LDAPControl **ctrls,
  unsigned long *result, char **attribute );
```

Parameters

This function has the following parameters:

Table 18-101 ldap_parse_sort_control() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`ctrls`	An array of controls returned by the server. You obtain these controls by calling the `ldap_parse_result()` function on the set of results returned by the server.
`result`	Pointer to the sort control result code retrieved by this function.
`attribute`	If the sorting operation fails, the function sets this to point to the name of the attribute that caused the failure. When done, you can free this by calling the `ldap_memfree()` function.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function (for example, if the LDAP connection is not valid).
- LDAP_NO_MEMORY if memory cannot be allocated to decode the control returned by the server.
- LDAP_DECODING_ERROR if an error occurred when BER decoding the control.
- LDAP_CONTROL_NOT_FOUND if no control can be found in the response returned from the server.

Description

Call the ldap_parse_sort_control() function as part of the process of retrieving sorted search results from a server.

Call this function after receiving the results (call ldap_result() to get the results) and parsing the server controls from the results (call ldap_parse_result() to get the controls from the results).

See Also

ldap_create_sort_control().

ldap_parse_virtuallist_control()

Parses the result returned from a search operation that used a server control for virtual list views.

This function implements an extension to the LDAPv3 protocol. A virtual list view is an optional LDAP server feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension.

Syntax

```
#include <ldap.h>
int ldap_parse_virtuallist_control( LDAP *ld,
  LDAPControl **ctrls, unsigned long *target_posp,
  unsigned long *list_sizep, int *errcodep );
```

Parameters

This function has the following parameters:

Table 18-102 ldap_parse_virtuallist_control() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`ctrls`	An array of controls returned by the server. You obtain these controls by calling the `ldap_parse_result()` function on the set of results returned by the server.
`target_posp`	Pointer to an unsigned long that is set by the function. The function sets this to the index or offset of the selected entry in the list of entries.
`list_sizep`	Pointer to an unsigned long that is set by the function. The function sets this to the number of entries in the total number of entries in the entire list (not just the subset).
`errcodep`	Pointer to the sort control result code retrieved by this function.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function (for example, if the LDAP connection is not valid).
- `LDAP_NO_MEMORY` if memory cannot be allocated to decode the control returned by the server.
- `LDAP_DECODING_ERROR` if an error occurred when BER decoding the control.
- `LDAP_CONTROL_NOT_FOUND` if no control can be found in the response returned from the server.
- `LDAP_NOT_SUPPORTED` if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client.”)

Description

Call the `ldap_parse_virtuallist_control()` function as part of the process of retrieving a subset of entries from a list when working with a virtual list view box.

Call this function after receiving the results (call `ldap_result()` to get the results) and parsing the server controls from the results (call `ldap_parse_result()` to get the controls from the results).

This function implements an extension to the LDAPv3 protocol. This control is supported by the Netscape Directory Server, version 4.0 and later; and all iPlanet Directory Server versions. For information on determining if a server supports this or other LDAPv3 controls, see "Determining If the Server Supports LDAPv3," on page 211.

For more information about this control, see "Using the Virtual List View Control."

See Also

`ldap_create_virtuallist_control()`.

ldap_perror()

The `ldap_perror()` function prints the last LDAP error message to standard output. The routine precedes this error message with the text you supply as the `s` parameter. For more information, see "Printing Out Error Messages."

Syntax

```
#include <ldap.h>
void ldap_perror( LDAP *ld, const char *s );
```

Parameters

This function has the following parameters:

Table 18-103 ldap_perror() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`s`	Text to print out before printing the error message.

Example

The following section of code prints out an error message if the search operation cannot complete successfully.

Code Example 18-39 ldap_perror() code example

```
...
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
  get_attr, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
...
```

See Also

`ldap_get_lderrno()`, `ldap_err2string()`, `ldap_result2error()`, `ldap_set_lderrno()`.

ldap_rename()

Changes the DN of an entry in the directory asynchronously.

Syntax

```
#include <ldap.h>
int ldap_rename( LDAP *ld, const char *dn, const char *newrdn,
  const char *newparent, int deleteoldrdn,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-104 ldap_rename() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to rename.
`newrdn`	New relative distinguished name (RDN) to assign to the entry.
`newparent`	DN of the new parent entry you want to move the entry under. Pass `NULL` if you do not want to move the entry to a different location in the directory tree.
`deleteoldrdn`	If this is a non-zero value, the old RDN is not retained as a value in the entry. If 0, the old RDN is retained as an attribute in the entry.

Table 18-104 ldap_rename() function parameters

`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`msgidp`	Pointer to an integer that will be set to the message ID of the LDAP operation.Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call the `ldap_result()` and `ldap_parse_result()` functions.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The `ldap_rename()` changes the distinguished name (DN) of an entry in the directory asynchronously and allows you to move the entry under a different parent entry in the directory tree.

This function is a new version of the `ldap_modrdn2()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_modrdn2()`.

You can specify whether or not the old RDN is retained as an attribute of the entry. Use the `deleteoldrdn` argument to do this. Suppose an entry has the following values for the `cn` attribute:

```
cn: Barbara Jensen
cn: Babs Jensen
```

If you change the RDN to "`cn=Barbie Jensen`" and pass "`1`" as `deleteoldrdn`, the resulting entry has the following values:

```
cn: Barbie Jensen
cn: Babs Jensen
```

If instead you pass `0` as `deleteoldrdn`, the "Barbara Jensen" value is not removed from the entry:

```
cn: Barbie Jensen
cn: Babs Jensen
cn: Barbara Jensen
```

`ldap_rename()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_rename_s()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP rename operation, you need to call the `ldap_result()` function and the `ldap_parse_result()` function. (See "Calling Asynchronous Functions" for details.) For a list of possible result codes for an LDAP rename operation, see the result code documentation for the `ldap_rename_s()` function.

For more information on changing the DN of an entry, see "Changing the DN of an Entry."

Example

See the example under "Example: Renaming an Entry in the Directory (Asynchronous)."

See Also

`ldap_rename_s(), ldap_result(), ldap_parse_result().`

ldap_rename_s()

Changes the DN of an entry in the directory synchronously.

Syntax

```
#include <ldap.h>
int ldap_rename_s( LDAP *ld, const char *dn, const char *newrdn,
  const char *newparent, int deleteoldrdn,
  LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-105 ldap_rename_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the entry to modify.
`newrdn`	New relative distinguished name (RDN) to assign to the entry.
`newparent`	DN of the new parent entry you want to move the entry under. Pass `NULL` if you do not want to move the entry to a different location in the directory tree.
`deleteoldrdn`	If this is a non-zero value, the old RDN is not retained as a value in the modified entry. If 0, the old RDN is retained as an attribute in the modified entry.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.

- LDAP_NOT_SUPPORTED if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client.”)

The following result codes can be returned by the Directory Server when processing an LDAP modify DN request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- LDAP_OPERATIONS_ERROR may be sent by the Directory Server for general errors encountered by the server when processing the request.
- LDAP_PROTOCOL_ERROR if the modify RDN request did not comply with the LDAP protocol. The Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
    - The server encountered an error when decoding your client’s BER-encoded request.
    - The RDN specified by the newrdn argument is not a valid RDN.
    - Your LDAP client has not specified that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client.”)
    - The DN specified by the newparent argument is not a valid DN.
- LDAP_UNWILLING_TO_PERFORM may be sent by the Directory Server in the following situations:
    - The entry to be renamed is a DSE (DSA-specific entry, where DSA is the Directory Server Agent).
    - You have specified a value for the newparent argument and the server does not support the ability to move an entry under another parent entry in the directory tree.
    - The server’s database is read-only.
- LDAP_NO_SUCH_OBJECT may be sent by the Directory Server if the entry that you want modified does not exist and if no referral URLs are available.
- LDAP_REFERRAL may be sent by the Directory Server if the DN specified by the dn argument identifies an entry not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is uid=bjensen, ou=European Sales, o=airius.com, all entries under ou=European Sales might be handled by a different directory server.)

- LDAP_NOT_ALLOWED_ON_NONLEAF may be sent by the Directory Server if the entry that you want renamed has entries beneath it in the directory tree (in other words, if this entry is a parent entry to other entries).
- LDAP_INSUFFICIENT_ACCESS may be sent by the Directory Server if the DN that your client is authenticated as does not have the access rights to write to the entry.
- LDAP_ALREADY_EXISTS may be sent by the Directory Server if the new DN identifies an entry that already exists in the directory (for example, if you want to change the DN of an entry to uid=bjensen, ou=People, o=airius.com but an entry with that DN already exists).

Note that the Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The ldap_rename_s() changes the DN of an entry in the directory synchronously and allows you to move the entry under a different parent entry in the directory tree.

This function is a new version of the ldap_modrdn2_s() function. If you are writing a new LDAP client, you should call this function instead of ldap_modrdn2_s().

You can specify whether or not the old RDN is retained as an attribute of the entry. Use the deleteoldrdn argument to do this. Suppose an entry has the following values for the cn attribute:

```
cn: Barbara Jensen
cn: Babs Jensen
```

If you change the RDN to cn=Barbie Jensen and pass 1 as deleteoldrdn, the resulting entry has the following values:

```
cn: Barbie Jensen
cn: Babs Jensen
```

If instead you pass 0 as deleteoldrdn, the "Barbara Jensen" value is not removed from the entry:

```
cn: Barbie Jensen
cn: Babs Jensen
cn: Barbara Jensen
```

The function ldap_rename_s() is synchronous; it directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function ldap_rename() instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For more information on changing the DN of an entry, see "Changing the DN of an Entry."

Example

See the example under "Example: Renaming an Entry in the Directory (Synchronous)."

See Also

ldap_rename().

ldap_result()

The function ldap_result() waits for and returns the result of an LDAP operation initiated by one of the asynchronous LDAP API functions (for example, ldap_search(), ldap_add(), ldap_modify()).

To identify the operation that you want to check, pass the message ID of the operation. (Asynchronous functions return a unique message ID that you can pass to the ldap_result() function.)

For more information, see "Getting the Server Response."

Syntax

```
#include <ldap.h>
int ldap_result( LDAP *ld, int msgid, int all,
  struct timeval *timeout, LDAPMessage **result );
```

Parameters

This function has the following parameters:

Table 18-106 ldap_result() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
msgid	Message ID of the operation for which you want the results. To check any operation, pass LDAP_RES_ANY as the value of this parameter.

Table 18-106 ldap_result() function parameters

`all`	Specifies how the results of a search are returned. This parameter can have the following values: • `0` specifies that the results are returned one entry at a time, using separate calls to `ldap_result()`. • A non-zero value specifies that all results are returned at the same time (after the final search result is obtained by the library).
`timeout`	Specifies a maximum interval to wait for the selection to complete. If timeout is a `NULL` pointer, the select blocks indefinitely. To effect a poll, the timeout parameter should be a non-`NULL` pointer, pointing to a zero-valued `timeval` structure.
`result`	Result of the operation. To interpret the results, pass this to the LDAP parsing routines, such as `ldap_result2error()`, `ldap_parse_result()`, and `ldap_first_entry()`.

Returns

One of the following values:

- `LDAP_RES_BIND` indicates that the `LDAPMessage` structure contains the result of an LDAP bind operation.
- `LDAP_RES_SEARCH_ENTRY` indicates that the `LDAPMessage` structure contains an entry found during an LDAP search operation.
- `LDAP_RES_SEARCH_REFERENCE` indicates that the `LDAPMessage` structure contains an LDAPv3 search reference (a referral to another LDAP server) found during an LDAP search operation.
- `LDAP_RES_SEARCH_RESULT` indicates that the `LDAPMessage` structure contains the result of an LDAP search operation.
- `LDAP_RES_MODIFY` indicates that the `LDAPMessage` structure contains the result of an LDAP modify operation.
- `LDAP_RES_ADD` indicates that the `LDAPMessage` structure contains the result of an LDAP add operation.
- `LDAP_RES_DELETE` indicates that the `LDAPMessage` structure contains the result of an LDAP delete operation.
- `LDAP_RES_MODDN` or `LDAP_RES_RENAME` indicates that the `LDAPMessage` structure contains the result of an LDAP modify DN operation.

- `LDAP_RES_COMPARE` indicates that the `LDAPMessage` structure contains the result of an LDAP compare operation.
- `LDAP_RES_EXTENDED` indicates that the `LDAPMessage` structure contains the result of an LDAPv3 extended operation.
- `-1` indicates that an error occurred. The error code is set in the `LDAP` structure. To get the error code, call the `ldap_get_lderrno()` function. (See Chapter 19, “Result Codes” for a complete listing of error codes.)
- `0` indicates that the operation has timed out.

Example

See the examples in the following sections:

- For an example of getting the results of an asynchronous LDAP add operation, see “Example: Adding an Entry to the Directory (Asynchronous).”
- For an example of getting the results of an asynchronous LDAP modify operation, see “Example: Modifying an Entry in the Directory (Asynchronous).”
- For an example of getting the results of an asynchronous LDAP delete operation, see “Example: Deleting an Entry from the Directory (Asynchronous).”
- For an example of getting the results of an asynchronous LDAP modify DN operation, see “Example: Renaming an Entry in the Directory (Asynchronous).”
- For an example of getting the results of an asynchronous LDAP search operation, see “Example: Searching the Directory (Asynchronous).”
- For an example of getting the results of an asynchronous LDAP bind operation, see “Performing an Asynchronous Authentication Operation.”

See Also

`ldap_add_ext()`, `ldap_compare_ext()`, `ldap_delete_ext()`, `ldap_modify_ext()`, `ldap_rename()`, `ldap_simple_bind()`, `ldap_url_search()`.

ldap_result2error()

This function is deprecated and is supported and documented in this release for backward compatibility only. In its place, you should use `ldap_parse_result()`.

`ldap_result2error()` returns the corresponding error code for a result produced by the `ldap_result()` and `ldap_search_s()` functions.

Syntax

```
#include <ldap.h>
int ldap_result2error( LDAP *ld, LDAPMessage *r, int freeit );
```

Parameters

This function has the following parameters:

Table 18-107 ldap_result2error() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`r`	Pointer to the `LDAPMessage` structure representing the results returned by the `ldap_result()` or `ldap_search()` function.
`freeit`	Specifies whether or not the result should be freed after the error code is extracted. The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values:

- If successful, sets the error code and other error information in the LDAP structure and returns the error code. (See Chapter 19, “Result Codes” for a complete listing of error codes.)
- If unsuccessful, returns `LDAP_PARAM_ERROR`.

Example

See the example under `ldap_result()`.

See Also

`ldap_parse_result()`, `ldap_get_lderrno()`, `ldap_err2string()`, `ldap_result()`, `ldap_set_lderrno()`.

ldap_sasl_bind()

Authenticates your client to an LDAP server using an SASL (Simple Authentication and Security Layer) mechanism. The LDAP server must support that SASL mechanism for authentication.

Syntax

```
#include <ldap.h>
int ldap_sasl_bind( LDAP *ld, const char *dn,
  const char *mechanism, const struct berval *cred,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
    int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-108 ldap_sasl_bind() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the user who wants to authenticate. For anonymous authentication, set this to NULL.
`mechanism`	Name of the SASL mechanism that you want to use for authentication.
`cred`	Pointer to the `berval` structure containing the credentials that you want to use for authentication.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`msgidp`	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call `ldap_result()` and `ldap_parse_sasl_bind_result()` functions.

Returns

One of the following values:

- `LDAP_SUCCESS` if the SASL bind request was sent successfully.

- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NOT_SUPPORTED` if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request to send to the server.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.

Description

The `ldap_sasl_bind()` function authenticates your client to an LDAP server by using a specified SASL mechanism. The LDAP server must support authentication through that SASL mechanism. (The Directory Server supports a server plug-in interface that you can use to add SASL support to the server. For details, see the *iPlanet Directory Server Plug-in Programmer's Guide*.)

`ldap_sasl_bind()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_sasl_bind_s()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP SASL bind operation, you need to call the `ldap_result()` function, the `ldap_parse_sasl_bind_result()` function, and the `ldap_get_lderrno()` function. (See "Performing an Asynchronous SASL Bind Operation" for details.) For a list of possible result codes for an LDAP SASL bind operation, see the result code documentation for the `ldap_sasl_bind_s()` function.

For additional information on authenticating through SASL mechanisms, see Chapter 13, "Using SASL Authentication."

See Also

`ldap_result()`, `ldap_parse_sasl_bind_result()`, `ldap_get_lderrno()`, `ldap_sasl_bind_s()`.

ldap_sasl_bind_s()

Authenticates your client to an LDAP server synchronously using an SASL (Simple Authentication and Security Layer) mechanism. The LDAP server must support that SASL mechanism for authentication.

Syntax

```
#include <ldap.h>
int ldap_sasl_bind_s( LDAP *ld, const char *dn,
  const char *mechanism, const struct berval *cred,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  struct berval **servercredp );
```

Parameters

This function has the following parameters:

Table 18-109 ldap_sasl_bind_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`dn`	Distinguished name (DN) of the user who wants to authenticate. For anonymous authentication, set this to NULL.
`mechanism`	Name of the SASL mechanism that you want to use for authentication.
`cred`	Pointer to the `berval` structure containing the credentials that you want to use for authentication.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`servercredp`	Pointer to a pointer to an `berval` structure containing any credentials returned by the server. When done, you can free this by calling the `ber_bvfree()` function.

Returns

One of the following values:

- `LDAP_SUCCESS` if your client authenticated successfully to the server.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.

- LDAP_NOT_SUPPORTED if your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request to send to the server.
- LDAP_DECODING_ERROR if an error occurred when the LDAP API library was decoding the BER-encoded results received from the server.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.

The following result codes can be returned by the Directory Server when processing an LDAP SASL bind request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- LDAP_OPERATIONS_ERROR may be sent by the Directory Server if the server cannot parse the LDAP controls that you are passing as arguments.
- LDAP_UNAVAILABLE_CRITICAL_EXTENSION may be sent by the Directory Server if you specify a critical LDAP control that is not supported by the server.
- LDAP_AUTH_METHOD_NOT_SUPPORTED can be set by the Directory Server if the SASL mechanism that you specify is not supported by the server (or if you specify an empty string as the SASL mechanism).
- LDAP_PROTOCOL_ERROR if the bind request sent by this function did not comply with the LDAP protocol. The Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
  - The server encountered an error when decoding your client's BER-encoded request.
  - Your LDAP client has not specified that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before sending a SASL bind request. (For details, see "Specifying the LDAP Version of Your Client.")

Note that the iPlanet Directory Server may send other result codes in addition to the codes described here (for example, `LDAP_NO_SUCH_OBJECT` if the DN in the original bind request does not exist or `LDAP_INVALID_CREDENTIALS` if the credentials in the original bind request were incorrect). In the Directory Server, the people deploying the server are responsible for implementing the authentication mechanisms for SASL authentication. Check with your server administrator for additional result codes returned to the client.

Description

The `ldap_sasl_bind_s()` function authenticates your client to an LDAP server by using a specified SASL mechanism. The LDAP server must support authentication through that SASL mechanism. (The Directory Server supports a server plug-in interface that you can use to add SASL support to the server. For details, see the *iPlanet Directory Server Plug-In Programmer's Guide*.)

After authenticating a client through SASL, an LDAP server can return a set of credentials in the results. The `servercredp` argument points to this value.

`ldap_sasl_bind_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_sasl_bind()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For additional information on authenticating through SASL mechanisms, see Chapter 13, "Using SASL Authentication."

See Also

`ldap_sasl_bind()`.

ldap_search()

Searches the directory asynchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_search( LDAP *ld, const char *base, int scope,
  const char* filter, char **attrs, int attrsonly );
```

Parameters

This function has the following parameters:

Table 18-110 ldap_search() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`base`	Distinguished name (DN) of the entry that serves as the starting point for the search. For example, setting `base` to `"o=airius.com"` restricts the search to entries at airius.com.
`scope`	Scope of the search, which can be one of the following values: • `LDAP_SCOPE_BASE` searches the entry specified by `base`. • `LDAP_SCOPE_ONELEVEL` searches all entries one level beneath the entry specified by `base`. • `LDAP_SCOPE_SUBTREE` searches the entry specified by `base` and all entries at all levels beneath the entry specified by `base`.
`filter`	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: • (*attributetype*=*attributevalue*) For details on the syntax for filters, see "Specifying a Search Filter."
`attrs`	A `NULL`-terminated array of attribute types to return from entries that match filter. If you specify a `NULL`, all attributes will be returned.
`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.

Returns

Returns the message ID of the `ldap_search()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()` For a list of possible result codes for an LDAP search operation, see the result code documentation for the `ldap_search_ext_s()` function.

Description

The `ldap_search()` function searches the directory asynchronously.

A newer version of this function, `ldap_search_ext()`, is available in this release of the LDAP API. `ldap_search()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext()` instead of `ldap_search()`.

If you want more information on `ldap_search()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code searches the directory.

Code Example 18-40 ldap_search() code example

```
#include "examples.h"

static void do_other_work();
unsigned long global_counter = 0;

int
main( int argc, char **argv )
{
  LDAP *ld;
  LDAPMessage *result, *e;
  BerElement *ber;
  char *a, *dn;
  char **vals;
  int i, rc, finished, msgid;
  int num_entries = 0;
  struct timeval zerotime;

  zerotime.tv_sec = zerotime.tv_usec = 0L;

/* get a handle to an LDAP connection */
if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}
/* authenticate to the directory as nobody */
if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_simple_bind_s" );
  return( 1 );
}
/* search for all entries with surname of Jensen */
if (( msgid = ldap_search( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE,
      MY_FILTER, NULL, 0 )) == -1 ) {
  ldap_perror( ld, "ldap_search" );
  return( 1 );
}

/* Loop, polling for results until finished */
finished = 0;
while ( !finished ) {
  /*
```

Code Example 18-40 ldap_search() code example

```
* Poll for results.   We call ldap_result with the "all" parameter
* set to zero. This causes ldap_result() to return exactly one
* entry if at least one entry is available. This allows us to
* display the entries as they are received.
*/
result = NULL;
rc = ldap_result( ld, msgid, 0, &zerotime, &result );
switch ( rc ) {
case -1:
  /* some error occurred */
  ldap_perror( ld, "ldap_result" );
  return( 1 );
case 0:
/* Timeout was exceeded. No entries are ready for retrieval. */
  if ( result != NULL ) {
    ldap_msgfree( result );
  }
  break;
default:
/*
* Either an entry is ready for retrieval, or all entries have
* been retrieved.
*/
  if (( e = ldap_first_entry( ld, result )) == NULL ) {
    /* All done */
    finished = 1;
    if ( result != NULL ) {
      ldap_msgfree( result );
    }
    continue;
    }
  /* for each entry print out name + all attrs and values */
  num_entries++;
  if (( dn = ldap_get_dn( ld, e )) != NULL ) {
    printf( "dn: %s\n", dn );
    ldap_memfree( dn );
  }
  for ( a = ldap_first_attribute( ld, e, &ber );
   a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
    if (( vals = ldap_get_values( ld, e, a )) != NULL ) {
      for ( i = 0; vals[ i ] != NULL; i++ ) {
        printf( "%s: %s\n", a, vals[ i ] );
      }
    ldap_value_free( vals );
    }
    ldap_memfree( a );
  }
  if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
  }
  printf( "\n" );
  ldap_msgfree( result );
}
/* Do other work here while you are waiting... */
do_other_work();
```

Code Example 18-40 ldap_search() code example

```
}

/* All done. Print a summary. */
printf( "%d entries retrieved. I counted to %ld "
      "while waiting.\n", num_entries, global_counter );
ldap_unbind( ld );
return( 0 );
}

/*
 * Perform other work while polling for results. */
static void
do_other_work()
{
    global_counter++;
}
```

See Also

`ldap_search_ext()`.

ldap_search_ext()

Searches the directory asynchronously.

Syntax

```
#include <ldap.h>
int ldap_search_ext( LDAP *ld, const char *base, int scope,
  const char *filter, char **attrs, int attrsonly,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  struct timeval *timeoutp, int sizelimit, int *msgidp );
```

Parameters

This function has the following parameters:

Table 18-111 ldap_sort_ext() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`base`	Distinguished name (DN) of the entry that serves as the starting point for the search. For example, setting `base` to `"o=airius.com"` restricts the search to entries at `airius.com`.

Table 18-111 ldap_sort_ext() function parameters

`scope`	Scope of the search, which can be one of the following values: • `LDAP_SCOPE_BASE` searches the entry specified by `base`. • `LDAP_SCOPE_ONELEVEL` searches all entries one level beneath the entry specified by `base`. • `LDAP_SCOPE_SUBTREE` searches the entry specified by `base` and all entries at all levels beneath the entry specified by `base`.
`filter`	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: • (*attributetype*=*attributevalue*) For details on the syntax for filters, see "Specifying a Search Filter."
`attrs`	A `NULL`-terminated array of attribute types to return from entries that match filter. If you specify a `NULL`, all attributes will be returned.
`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`timeoutp`	Pointer to a `timeval` structure specifying the maximum time to wait for the results of the search. Pass `NULL` to use the default time limit for the current connection. To specify an infinite time limit, set the `tv_sec` and `tv_usec` fields in the `timeval` structure to 0.
`sizelimit`	Maximum number of results to return in the search. Pass `-1` to use the default size limit for the current connection.
`msgidp`	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call the `ldap_result()` and `ldap_parse_result()` functions.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.")

Description

The `ldap_search_ext()` function searches the directory for matching entries asynchronously.

This function is a new version of the `ldap_search()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_search()`.

You can use this function to pass LDAP server controls to the server if you want the server to sort the results or if you want to request a persistent search. (See `ldap_create_sort_control()` and `ldap_create_persistentsearch_control()` for more information.)

`ldap_search_ext()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_search_ext_s()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

In order to get the results of the LDAP search operation, you need to call the `ldap_result()` function and the `ldap_parse_result()` function. (See "Getting Results Asynchronously" for details.) For a list of possible result codes for an LDAP search operation, see the result code documentation for the `ldap_search_ext_s()` function.

For more information on searching the directory, see Chapter 6, "Searching the Directory."

Example

See the example under “Example: Searching the Directory (Asynchronous).”

See Also

`ldap_search_ext_s(), ldap_result(), ldap_parse_result().`

ldap_search_ext_s()

Searches the directory synchronously.

Syntax

```
#include <ldap.h>
int ldap_search_ext_s( LDAP *ld, const char *base, int scope,
  const char *filter, char **attrs, int attrsonly,
  LDAPControl **serverctrls, LDAPControl **clientctrls,
  struct timeval *timeoutp, int sizelimit, LDAPMessage **res );
```

Parameters

This function has the following parameters:

Table 18-112 ldap_search_ext_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`base`	Distinguished name (DN) of the entry that serves as the starting point for the search. For example, setting `base` to `"o=airius.com"` restricts the search to entries at `airius.com`.
`scope`	Scope of the search, which can be one of the following values: • `LDAP_SCOPE_BASE` searches the entry specified by `base`. • `LDAP_SCOPE_ONELEVEL` searches all entries one level beneath the entry specified by `base`. • `LDAP_SCOPE_SUBTREE` searches all entries at all levels beneath the entry specified by `base`.
`filter`	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: • (*attributetype=attributevalue*) For details on the syntax for filters, see “Specifying a Search Filter.”
`attrs`	A `NULL`-terminated array of attribute types to return from entries that match filter. If you specify a `NULL`, all attributes will be returned.

Table 18-112 ldap_search_ext_s() function parameters

`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.
`serverctrls`	Pointer to an array of `LDAPControl` structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify `NULL` for this argument.
`clientctrls`	Pointer to an array of `LDAPControl` structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify `NULL` for this argument.
`timeoutp`	Pointer to a `timeval` structure specifying the maximum time to wait for the results of the search.
`sizelimit`	Maximum number of results to return in the search.
`res`	Results of the search (when the call is completed).

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client.”)

- LDAP_FILTER_ERROR if an error occurred when parsing and BER-encoding the search filter specified by the filter argument.
- LDAP_TIMEOUT if the search exceeded the time specified by the timeoutp argument.

The following result codes can be returned by the iPlanet Directory Server when processing an LDAP search request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- LDAP_OPERATIONS_ERROR may be sent by the Directory Server for general errors encountered by the server when processing the request.
- LDAP_PROTOCOL_ERROR if the search request did not comply with the LDAP protocol. The Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
  - The server encountered an error when decoding your client's BER-encoded request.
  - The search request received by the server specifies an unknown search scope or filter type.
  - Your LDAP client attempts to use an extensible search filter and has not specified that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before sending the request. (For details, see "Specifying the LDAP Version of Your Client.")
  - The server encountered an error when attempting to sort the search results or when attempting to send sorted results.
- LDAP_INVALID_SYNTAX may be sent by the Directory Server if your LDAP client specified a substring filter containing no value for comparison.
- LDAP_NO_SUCH_OBJECT may be sent by the Directory Server if the entry specified by the base argument does not exist and if no referral URLs are available.
- LDAP_REFERRAL may be sent by the Directory Server if the entry specified by the base argument is not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is uid=bjensen, ou=European Sales, o=airius.com, all entries under ou=European Sales might be handled by a different directory server.)
- LDAP_TIMELIMIT_EXCEEDED may be sent by the Directory Server if the search exceeded the maximum time specified by the timeoutp argument.

- `LDAP_SIZELIMIT_EXCEEDED` may be sent by the Directory Server if the search found more results than the maximum number of results specified by the `sizelimit` argument.
- `LDAP_ADMINLIMIT_EXCEEDED` may be sent by the Directory Server if the search found more results than the limit specified by the `lookthroughlimit` directive in the `slapd.conf` configuration file. (If not specified in the configuration file, the limit is 5000.)

Note that the Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The `ldap_search_ext_s()` searches the directory for matching entries synchronously.

You can use this function to pass LDAP server controls to the server if you want the server to sort the results or if you want to request a persistent search. (See `ldap_create_sort_control()` and `ldap_create_persistentsearch_control()` for more information.)

This function is a new version of the `ldap_search_s()` function. If you are writing a new LDAP client, you should call this function instead of `ldap_search_s()`.

The function `ldap_search_ext_s()` is synchronous; it directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_search_ext()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For more information on searching the directory, see Chapter 6, "Searching the Directory."

Example

See the example under "Example: Searching the Directory (Synchronous)."

See Also

`ldap_search_ext()`.

ldap_search_s()

Searches the directory synchronously.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_search_s( LDAP *ld, const char *base, int scope,
  const char* filter, char **attrs, int attrsonly,
  LDAPMessage **res );
```

Parameters

This function has the following parameters:

Table 18-113 ldap_search_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`base`	Distinguished name (DN) of the entry that serves as the starting point for the search. For example, setting `base` to `"o=airius.com"` restricts the search to entries at `airius.com`.
`scope`	Scope of the search, which can be one of the following values: • `LDAP_SCOPE_BASE` searches the entry specified by `base`. • `LDAP_SCOPE_ONELEVEL` searches all entries one level beneath the entry specified by `base`. • `LDAP_SCOPE_SUBTREE` searches all entries at all levels beneath the entry specified by `base`.
`filter`	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: • (*attributetype=attributevalue*) For details on the syntax for filters, see "Specifying a Search Filter."
`attrs`	A `NULL`-terminated array of attribute types to return from entries that match filter. If you specify a `NULL`, all attributes will be returned.
`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.
`res`	Results of the search (when the call is completed).

Returns

For a list of possible result codes for an LDAP search operation, see the result code documentation for the `ldap_search_ext_s()` function.

Description

The `ldap_search_s()` function searches the directory for matching entries.

A newer version of this function, `ldap_search_ext_s()`, is available in this release of the LDAP API. `ldap_search_s()` (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext_s()` instead of `ldap_search_s()`.

If you want more information on `ldap_search_s()`, refer to the *LDAP C SDK 1.0 Programmer's Guide*.

Example

The following section of code searches the directory for all people whose surname (last name) is "`Jensen`".

Code Example 18-41 ldap_search_s() code example

```
#include "examples.h"

int main( int argc, char **argv )
{
  LDAP *ld;
  LDAPMessage *result, *e;
  BerElement *ber;
  char *a, *dn;
  char **vals;
  int i;
  /* get a handle to an LDAP connection */
  if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
  /* authenticate to the directory as nobody */
  if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_simple_bind_s" );
    return( 1 );
  }
  /* search for all entries with surname of Jensen */
  if ( ldap_search_s( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE,
    MY_FILTER, NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
  }
  /* for each entry print out name + all attrs and values */
  for ( e = ldap_first_entry( ld, result ); e != NULL;
   e = ldap_next_entry( ld, e ) ) {
    if ( (dn = ldap_get_dn( ld, e )) != NULL ) {
```

Code Example 18-41 ldap_search_s() code example

```
      printf( "dn: %s\n", dn );
      ldap_memfree( dn );
    }
    for ( a = ldap_first_attribute( ld, e, &ber );
     a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
      if ((vals = ldap_get_values( ld, e, a)) != NULL ) {
        for ( i = 0; vals[i] != NULL; i++ ) {
          printf( "%s: %s\n", a, vals[i] );
        }
        ldap_value_free( vals );
      }
      ldap_memfree( a );
    }
    if ( ber != NULL ) {
      ldap_ber_free( ber, 0 );
    }
    printf( "\n" );
  }
  ldap_msgfree( result );
  ldap_unbind( ld );
  return( 0 );
}
```

See Also

`ldap_search_ext_s()`.

ldap_search_st()

Searches the directory synchronously within a specified time limit.

Note that this is an older function that is included in the LDAP API for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_search_st( LDAP *ld, const char *base, int scope,
  const char* filter, char **attrs, int attrsonly,
  struct timeval *timeout, LDAPMessage **res );
```

Parameters

This function has the following parameters:

Table 18-114 ldap_search_st() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`base`	Distinguished name (DN) of the entry that serves as the starting point for the search. For example, setting `base` to `"o=airius.com"` restricts the search to entries at `airius.com`.
`scope`	Scope of the search, which can be one of the following values: • `LDAP_SCOPE_BASE` searches the entry specified by `base`. • `LDAP_SCOPE_ONELEVEL` searches all entries one level beneath the entry specified by `base`. • `LDAP_SCOPE_SUBTREE` searches all entries at all levels beneath the entry specified by `base`.
`filter`	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: • (*attributetype*=*attributevalue*) For details on the syntax for filters, see "Specifying a Search Filter."
`attrs`	A `NULL`-terminated array of attribute types to return from entries that match filter. If you specify a `NULL`, all attributes will be returned.
`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.
`timeout`	Maximum time to wait for the results of the search.
`res`	Results of the search (when the call is completed).

Returns

For a list of possible result codes for an LDAP search operation, see the result code documentation for the `ldap_search_ext_s()` function.

Description

The ldap_search_st() function searches the directory for matching entries. The ldap_search_st() function works like the ldap_search_s() function and lets you set a timeout period for the search.

A newer version of this function, ldap_search_ext_s(), is available in this release of the LDAP API. ldap_search_st() (the older version of the function) is included only for backward-compatibility. If you are writing a new LDAP client, use ldap_search_ext_s() instead of ldap_search_st().

If you want more information on ldap_search_st(), refer to the *LDAP C SDK 1.0 Programmer's Guide*.

See Also

ldap_search_ext_s().

ldap_set_filter_additions()

The ldap_set_filter_additions() function sets a prefix to be prepended and a suffix to be appended to all filters returned by the ldap_getfirstfilter() and ldap_getnextfilter() function calls.

Syntax

```
#include <ldap.h>
int ldap_set_filter_additions( LDAPFiltDesc *lfdp, char *prefix,
  char *suffix );
```

Parameters

This function has the following parameters:

Table 18-115 ldap_set_filter_additions() function parameters

lfdp	Pointer to an LDAPFiltDesc structure.
prefix	Prefix to prepend to all filters. If NULL, no prefix is prepended.
suffix	Suffix to append to all filters. If NULL, no suffix is appended.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns an LDAP error code. (See Chapter 19, "Result Codes" for a complete listing.)

Example

The following section of code loads the filter configuration file named `myfilters.conf` into memory and adds the prefix "`(&(objectClass=person)`" and the suffix ")" to each filter retrieved:

Code Example 18-42 ldap_set_filter_additions() code example

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
char *prefix = "(&(objectClass=person)";
char *suffix = ")";
int rc;
...
lfdp = ldap_init_getfilter( filter_file );
rc = ldap_set_filter_additions( ldfp, prefix, suffix );
if ( rc != LDAP_SUCCESS ) {
  printf( "Error setting filter prefix and suffix\n");
  return( rc );
}
...
```

See Also

`ldap_getfirstfilter(), ldap_getnextfilter().`

ldap_setfilteraffixes()

The `ldap_setfilteraffixes()` function is a deprecated function. Use the `ldap_set_filter_additions()` function instead.

ldap_set_lderrno()

The `ldap_set_lderrno()` function sets an error code and information about an error in an LDAP structure. You can call this function to set error information that will be retrieved by subsequent `ldap_get_lderrno()` function calls.

For more information, see “Setting Error Codes.”

Syntax

```
#include <ldap.h>
int ldap_set_lderrno( LDAP *ld, int e, char *m, char *s );
```

Parameters

This function has the following parameters:

Table 18-116 ldap_set_lderrno() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
e	The error code that you want to set.
m	In the event that an entry for a specified DN cannot be found, you should set this parameter to the portion of the DN that identifies an existing entry. (See "Receiving the Portion of the DN Matching an Entry" for details.)
s	The text of the error message that you want associated with this error code.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns an LDAP error code. (See Chapter 19, "Result Codes" for a complete listing.)

Example

The following section of code attempts to perform an operation. If the operation fails, the LDAP_PARAM_ERROR error code is placed in the LDAP structure.

Code Example 18-43 ldap_set_lderrno() code example

```
#include <ldap.h>
int rc;
char *errmsg = "Invalid parameter";
...
if ( ldap_my_function() != LDAP_SUCCESS ) {
  rc = ldap_set_lderrno( ld, LDAP_PARAM_ERROR, NULL, errmsg );
  if ( rc != LDAP_SUCCESS ) {
    printf( "Error: %d\nError code could not be set.\n", rc );
  }
  return( rc );
}
...
```

See Also

ldap_err2string(), ldap_perror(), ldap_result2error().

ldap_set_option()

The function ldap_set_option() sets session preferences in the LDAP structure. For more information on the options you can set, see “Setting Preferences.”

Syntax

```
#include <ldap.h>
int ldap_set_option( LDAP *ld, int option, const void *optdata );
```

Parameters

This function has the following parameters:

Table 18-117 ldap_set_option() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server. If NULL, you are setting the default options that will apply to any new LDAP connection handles that are subsequently created.
option	Option that you want to set.
optdata	Pointer to the value of the option that you want to set.

The option parameter can have one of the following values:

Table 18-118 Options for ldap_set_option() function

LDAP_OPT_CLIENT_CONTROLS	Pointer to an array of LDAPControl structures representing the LDAPv3 client controls you want sent with every request by default. The data type for the optdata parameter is (LDAPControl ***).

Table 18-118 Options for ldap_set_option() function

LDAP_OPT_DEREF	Determines how aliases are handled during a search. The data type for the `optdata` parameter is `(int *)`. `optdata` can be one of the following values: • LDAP_DEREF_NEVER specifies that aliases are never dereferenced. • LDAP_DEREF_SEARCHING specifies that aliases are dereferenced when searching under the base object (but not when finding the base object). • LDAP_DEREF_FINDING specifies that aliases are dereferenced when finding the base object (but not when searching under the base object). • LDAP_DEREF_ALWAYS specifies that aliases are always dereferenced when finding the base object and searching under the base object.
LDAP_OPT_DNS_FN_PTRS	Lets you use alternate DNS functions for getting the host entry of an LDAP server. The data type for the `optdata` parameter is `(struct ldap_dns_fns *)`.
LDAP_OPT_EXTRA_THREAD_FN_PTRS	Lets you specify the locking and semaphore functions called when getting results from the server. (See Chapter 16, "Writing Multithreaded Clients" for details.) The data type for the `optdata` parameter is `(struct ldap_extra_thread_fns *)`.
LDAP_OPT_IO_FN_PTRS	Lets you use alternate communication stacks. The data type for the `optdata` parameter is `(struct ldap_io_fns *)`.
LDAP_OPT_MEMALLOC_FNS_PTRS	Lets you replace the standard function used for memory management (for example `malloc()`, `calloc()`, `realloc()`, and `free()`) with your own function. See the prototype for the `ldap_memalloc_fns` callback structure for the details on the structure that you can set.

Table 18-118 Options for ldap_set_option() function

LDAP_OPT_PROTOCOL_VERSION	Version of the protocol supported by your client. The data type for the `optdata` parameter is `(int *)`. You can specify either `LDAP_VERSION2` or `LDAP_VERSION3`. If no version is set, the default is `LDAP_VERSION2`. In order to use LDAPv3 features, you need to set the protocol version to `LDAP_VERSION3`.
LDAP_OPT_REBIND_ARG	Lets you set the last argument passed to the routine specified by `LDAP_OPT_REBIND_FN`. You can also set this option by calling the `ldap_set_rebind_proc()` function. The data type for the `optdata` parameter is `(void *)`.
LDAP_OPT_REBIND_FN	Lets you set the routine to be called when you need to authenticate a connection with another LDAP server (for example, during the course of a referral). You can also set this option by calling the `ldap_set_rebind_proc()` function. The data type for the `optdata` parameter is `(LDAP_REBINDPROC_CALLBACK *)`.
LDAP_OPT_RECONNECT	If the connection to the server is lost, determines whether or not the same connection handle should be used to reconnect to the server. The data type for the `optdata` parameter is `(int *)`. `optdata` can be one of the following values: • `LDAP_OPT_ON` specifies that the same connection handle can be used to reconnect to the server. • `LDAP_OPT_OFF` specifies that you want to create a new connection handle to connect to the server. By default, this option is off. For details, see “Handling Failover.”

Table 18-118 Options for ldap_set_option() function

LDAP_OPT_REFERRALS	Determines whether or not the client should follow referrals. The data type for the optdata parameter is (int *). optdata can be one of the following values: • LDAP_OPT_ON specifies that the client should follow referrals. • LDAP_OPT_OFF specifies that the client should not follow referrals. By default, the client follows referrals.
LDAP_OPT_REFERRAL_HOP_LIMIT	Maximum number of referrals the client should follow in a sequence (in other words, the client can only be referred this number of times before it gives up). The data type for the optdata parameter is (int *). By default, the maximum number of referrals that the client can follow in a sequence is 5 for the initial connection. Note that this limit does not apply to individual requests that generate multiple referrals in parallel.
LDAP_OPT_RESTART	If there is a fatal error on connection, this option resets the connection to an initial unconnected known state.
LDAP_OPT_SERVER_CONTROLS	Pointer to an array of LDAPControl structures representing the LDAPv3 server controls you want sent with every request by default. The data type for the optdata parameter is (LDAPControl ***).
LDAP_OPT_SIZELIMIT	Maximum number of entries that should be returned by the server in search results. The data type for the optdata parameter is (int *).
LDAP_OPT_SSL	Determines whether or not SSL is enabled. The data type for the optdata parameter is (int *). optdata can be one of the following values: • LDAP_OPT_ON specifies that SSL is enabled. • LDAP_OPT_OFF specifies that SSL is disabled.
LDAP_OPT_THREAD_FN_PTRS	Lets you specify the thread function pointers. See Chapter 16, “Writing Multithreaded Clients” for details. The data type for the optdata parameter is (struct ldap_thread_fns *).

Table 18-118 Options for ldap_set_option() function

LDAP_OPT_TIMELIMIT	Maximum number of seconds that should be spent by the server when answering a search request. The data type for the optdata parameter is (int *).
LDAP_X_OPT_EXTIO_FN_PTRS	Lets you specify the extended I/O callback functions which were introduced with version 4.0 of the LDAP SDK for C.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- -1 if unsuccessful.

Example

The following section of code sets the maximum number of entries returned in a search.

Code Example 18-44 ldap_set_option() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
int max_ret = 100, max_tim = 30;
char *host = "ldap.netscape.com";
...
/* Initialize a session with the LDAP server ldap.netscape.com:389 */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Set the maximum number of entries returned */
if (ldap_set_option( ld, LDAP_OPT_SIZELIMIT, &max_ret) != LDAP_SUCCESS) {
  ldap_perror( ld, "ldap_set_option" );
  return( 1 );
}
...
```

See Also

ldap_init(), ldap_get_option().

ldap_set_rebind_proc()

Sets the rebind function, which is the function called by your client to obtain authentication credentials when following a referral.

Syntax

```
#include <ldap.h>
void ldap_set_rebind_proc( LDAP *ld,
  LDAP_REBINDPROC_CALLBACK *rebindproc, void *arg );
```

Parameters

This function has the following parameters:

Table 18-119 ldap_set_rebind_proc() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`rebindproc`	Name of the rebind function.
`arg`	Pointer to an additional argument that you want to pass to the rebind function.

Description

Call `ldap_set_rebind_proc()` to specify the rebind function (the function called by the LDAP client when following a referral to a new LDAP server). This rebind function is responsible for obtaining the credentials used to authenticate to the new LDAP server.

For example, suppose LDAP server A sends a referral to your client. The referral points your client to LDAP server B. When automatically following the referral, your client calls the rebind function to obtain a DN and credentials; your client uses these to authenticate to server B.

By default, if you do not call `ldap_set_rebind_proc()` or if you pass NULL for the `rebindproc` argument, your client authenticates anonymously when following referrals.

The rebind function that you specify with `ldap_set_rebind_proc()` should have the following prototype:

```
int LDAP_CALL LDAP_CALLBACK rebindproc( LDAP *ld, char **dnp,
  char **passwdp, int *authmethodp, int freeit, void *arg );
```

(`LDAP_CALL` and `LDAP_CALLBACK` are used to set up calling conventions, such as Pascal calling conventions on Windows. These are defined in the `lber.h` header file.)

LDAP clients that are built with the Netscape LDAP SDK for C use this procedure when following referrals (the procedure explains what the rebind function is expected to do):

1. The LDAP server sends a referral back to the client.
2. The client calls the rebind function, passing `0` as the `freeit` argument.
3. The rebind function sets the `dnp`, `passwdp`, and `authmethodp` arguments to point to the following information:
   - The `dnp` argument is set to point to the DN to be used to authenticate to the new LDAP server.
   - The `passwdp` argument is set to point to the credentials for this DN.
   - The `authmethodp` argument is set to point to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).
4. If successful, the rebind function returns `LDAP_SUCCESS`, and referral processing continues. (If any other value is returned, referral processing stops, and that value is returned as the result code for the original LDAP request.)
5. The client gets the DN, credentials, and authentication method from the arguments of the rebind function and uses this information to authenticate to the new LDAP server.
6. The client calls the rebind function again, passing `1` as the `freeit` argument.
7. The rebind function frees any memory allocated earlier to specify the DN and credentials.

You need to write a rebind function that does the following:

- If `freeit` is `0`, set the following pointers:
  - Set `dnp` to point to the DN to be used for authentication.
  - Set `passwdp` to point to the credentials to be used for authentication.
  - Set `authmethodp` to point to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).

  You can make use of the `arg` argument, which is a pointer to the argument specified in the `ldap_set_rebind_proc()` function.

  If successful, return `LDAP_SUCCESS`. Otherwise, return the appropriate LDAP error code.
- If `freeit` is `1`, free any memory that you allocated to create the DN and credentials.

After you have defined this function, pass the function name to `ldap_set_rebind_proc()` to register your rebind function.

Note that in order to use the rebind function, the `LDAP_OPT_REFERRALS` option must be set to `LDAP_OPT_ON`, so that your client automatically follows referrals. This option is already set to `LDAP_OPT_ON` by default.

Example

The following example demonstrates how to write and register a rebind function.

Code Example 18-45 ldap_set_rebind_proc() code example

```
#include "ldap.h"
...
/* Declare your rebind function */
int rebindproc( LDAP *ld, char **dnp, char **passwdp, int *authmethodp, int
freeit, void *arg );
...
int main( int argc, char **argv )
{
  LDAP *ld;
  /* Additional argument to be passed to the rebind function */
  char *testarg = "cn=Directory Manager";
  /* Get a handle to an LDAP connection */
  if ( (ld = ldap_init( "directory.myhost.com", 389 )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
  }
  /* Specify the function used for reauthentication on referrals */
  ldap_set_rebind_proc( ld, rebindproc, (void *)testarg );
  /* Authenticate */
  if ( ldap_simple_bind_s( ld, "uid=bjensen,ou=People,o=airius.com",
      "hifalutin" ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_simple_bind_s" );
    return( 1 );
  }
  ...
  /* Your code to interact with the LDAP server */
  ...
}
...
/* rebindproc is the rebind function responsible for providing the DN,
  credentials, and authentication method used for authenticating the
  client to other Directory Servers.
  The function should set the following arguments:
  - dnp should point to the DN that will be used for authentication.
  - passwdp should point to the credentials used for authentication.
  - authmethodp should point to the method of authentication to be used
  (for example, LDAP_AUTH_SIMPLE).
  The function should return LDAP_SUCCESS if successful or an LDAP
  error code if an error occurs.
  In order to demonstrate how the freeit argument works, this example
```

Code Example 18-45 ldap_set_rebind_proc() code example

```
#include "ldap.h"
  uses strdup() to copy the DN and password.  You can also just copy
  string pointers if the DN and password are already available as
  global variables.
*/
int LDAP_CALL LDAP_CALLBACK rebindproc( LDAP *ld, char **dnp, char **passwdp,
int *authmethodp, int freeit, void *arg )
{
  printf( "Rebind function called.\n" );
  switch ( freeit ) {
  /* Your client calls the rebind function with freeit==1 when it needs
    to free any memory you've allocated. */
  case 1:
    printf( "\tFreeing memory.\n" );
    if ( dnp && *dnp ) {
      free( *dnp );
    }
    if ( passwdp && *passwdp ) {
      free( *passwdp );
    }
    break;
  /* Your client calls the rebind function with freeit==0 when it needs
  to get the DN, credentials, and authentication method. */
  case 0:
    printf( "\tGetting DN and credentials.\n" );
    *dnp = strdup( "uid=username,o=OtherServerSuffix" );
    *passwdp = strdup( "23skidoo" );
    *authmethodp = LDAP_AUTH_SIMPLE;
    break;
  default:
    printf( "\tUnknown value of freeit argument: %d\n", freeit );
    break;
  }
  /* If you successfully set the DN and credentials, you should return
    LDAP_SUCCESS.  (Any other return code will stop the client from
    automatically following the referral. */
  return LDAP_SUCCESS;
}
```

See Also

`ldap_simple_bind(), ldap_simple_bind_s().`

ldap_simple_bind()

Asynchronously authenticates your client to the LDAP server using a DN and a password.

Syntax

```
#include <ldap.h>
int ldap_simple_bind(LDAP *ld, const char *who,
  const char *passwd);
```

Parameters

This function has the following parameters:

Table 18-120 ldap_simple_bind() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`who`	Distinguished name (DN) of the user who wants to authenticate. For anonymous authentication, set this or the `passwd` argument to `NULL`.
`passwd`	Password of the user who wants to authenticate. For anonymous authentication, set this or the `who` argument to `NULL`.

Returns

Returns the message ID of the `ldap_simple_bind()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`.

Description

The `ldap_simple_bind()` function authenticates to the LDAP server. The function verifies that the password supplied for authentication matches the `userPassword` attribute of the given entry.

`ldap_simple_bind()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_simple_bind_s()` instead. (For more information on asynchronous and synchronous functions, see “Calling Synchronous and Asynchronous Functions.”)

Note that if you specify a DN but no password, your client will bind to the server anonymously. If you want a `NULL` password to be rejected as an incorrect password, you need to write code to perform the check before you call the `ldap_simple_bind()` function.

For additional information on authenticating to the LDAP server, see “Binding and Authenticating to an LDAP Server.”

Example

The following section of code calls `ldap_simple_bind()` to authenticate the user "Barbara Jensen" to the directory.

Code Example 18-46 ldap_simple_bind() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *host = "ldap.netscape.com";
char *dn = "uid=bjensen, ou=People, o=airius.com";
char *pw = "hifalutin";
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Initialize a session with the LDAP server ldap.netscape.com:389 */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}
/* Attempt to bind with the LDAP server */
msgid = ldap_simple_bind( ld, dn, pw );

/* Initialize the value returned by ldap_result() */
rc = 0;

/* While the operation is still running, do this: */
while ( rc == 0 ) {
  ... /* do other work while waiting */...

  /* Check the status of the LDAP operation */
  rc = ldap_result( ld, msgid, NULL, &zerotime, &result );
  switch( rc ) {
    /* If -1 was returned, an error occurred */
    case -1:
      ldap_perror( ld, "Error in results: " );
      return( 1 );
    /* If 0 was returned, the operation is still in progress */
    case 0:
      continue;
    /* If any other value is returned, assume we are done */
    default:
      /* Check if the "bind" operation was successful */
      if ( ldap_result2error( result ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "Error binding to server: " );
        return( 1 );
        }
  }
}
...
```

See Also

`ldap_simple_bind_s()`.

ldap_simple_bind_s()

Synchronously authenticates your client to the LDAP server using a DN and a password.

Syntax

```
#include <ldap.h>
int ldap_simple_bind_s( LDAP *ld, const char *who,
  const char *passwd );
```

Parameters

This function has the following parameters:

Table 18-121 ldap_simple_bind_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`who`	Distinguished name (DN) of the user who wants to authenticate. For anonymous authentication, set this or the `passwd` argument to `NULL`.
`passwd`	Password of the user who wants to authenticate. For anonymous authentication, set this or the `who` argument to `NULL`.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if any of the arguments are invalid.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.
- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.

- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see "Specifying the LDAP Version of Your Client."

The following result codes can be returned by the iPlanet Directory Server when processing an LDAP search request. Other LDAP servers may send these result codes under different circumstances or may send different result codes back to your LDAP client.

- `LDAP_OPERATIONS_ERROR` may be sent by the Directory Server for general errors encountered by the server when processing the request.
- `LDAP_PROTOCOL_ERROR` if the search request did not comply with the LDAP protocol. The Directory Server may set this error code in the results for a variety of reasons. Some of these reasons include:
  - The server encountered an error when decoding your client's BER-encoded request.
  - The search request received by the server specifies an unknown search scope or filter type.
  - Your LDAP client attempts to use an extensible search filter and has not specified that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before sending the request. For details, see "Specifying the LDAP Version of Your Client."
  - The server encountered an error when attempting to sort the search results or when attempting to send sorted results.
- `LDAP_INVALID_SYNTAX` may be sent by the Directory Server if your LDAP client specified a substring filter containing no value for comparison.
- `LDAP_NO_SUCH_OBJECT` may be sent by the Directory Server if the entry specified by the `base` argument does not exist and if no referral URLs are available.
- `LDAP_REFERRAL` may be sent by the Directory Server if the entry specified by the `base` argument is not handled by the current server and if referral URLs identify a different server to handle the entry. (For example, if the DN is `uid=bjensen, ou=European Sales, o=airius.com`, all entries under `ou=European Sales` might be handled by a different directory server.)
- `LDAP_TIMELIMIT_EXCEEDED` may be sent by the Directory Server if the search exceeded the maximum time specified by the `timeoutp` argument.

- LDAP_SIZELIMIT_EXCEEDED may be sent by the Directory Server if the search found more results than the maximum number of results specified by the sizelimit argument.
- LDAP_ADMINLIMIT_EXCEEDED may be sent by the Directory Server if the search found more results than the limit specified by the lookthroughlimit directive in the slapd.conf configuration file. (If not specified in the configuration file, the limit is 5000.)

Note that the iPlanet Directory Server may send other result codes in addition to the codes described here (for example, the server may have loaded a custom plug-in that returns other result codes).

Description

The ldap_simple_bind_s() function authenticates to the LDAP server. The function verifies that the password supplied for authentication matches the userPassword attribute of the given entry.

ldap_simple_bind_s() is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function ldap_simple_bind() instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

Note that if you specify a DN but no password, your client will bind to the server anonymously. If you want a NULL password to be rejected as an incorrect password, you need to write code to perform the check before you call the ldap_simple_bind_s() function.

For additional information on authenticating to the LDAP server, see "Binding and Authenticating to an LDAP Server."

Example

The following section of code uses the synchronous ldap_simple_bind_s() function to authenticate to the directory as the user "Barbara Jensen".

Code Example 18-47 ldap_simple_bind_s() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *host = "ldap.netscape.com";
char *dn = "uid=bjensen, ou=People, o=airius.com";
char *pw = "hifalutin";
...
```

Code Example 18-47 ldap_simple_bind_s() code example

```
#include <stdio.h>
/* Initialize a session with the LDAP server ldap.netscape.com:389 */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}
/* Attempt to bind with the LDAP server */
if ( ldap_simple_bind_s( ld, dn, pw ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "Authentication failed: " );
  return( 1 );
}
...
```

See Also

`ldap_simple_bind()`.

ldap_sort_entries()

The `ldap_sort_entries()` function sorts a chain of entries retrieved from an LDAP search call (`ldap_search_s()` or `ldap_result()`) either by distinguished name (DN) or by a specified attribute in the entries.

For additional information, see "Sorting the Search Results."

Syntax

```
#include <ldap.h>
int ldap_sort_entries( LDAP *ld, LDAPMessage *chain, char *attr,
  LDAP_CMP_CALLBACK *cmp );
```

Parameters

This function has the following parameters:

Table 18-122 ldap_sort_entires() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`chain`	Chain of entries returned by the `ldap_result()` or `ldap_search_s()` function.
`attr`	Attribute to use when sorting the results. To sort by distinguished name instead of by attribute, use `NULL`.
`cmp`	Comparison function used when sorting the values. For details, see "Sorting Entries by an Attribute."

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call ldap_get_lderrno( ). (See Chapter 19, “Result Codes” for a complete listing of error codes.)

Example

The following section of code sorts entries by the roomNumber attribute.

Code Example 18-48 ldap_sort_entries() code example

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
char *sortby = "roomNumber";
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter, NULL, 0,
&result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
}
/* Sort the results by room number, using strcasecmp */
if ( ldap_sort_entries( ld, &result, sortby, strcasecmp ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_sort_entries" );
  return( 1 );
}
...
```

See Also

ldap_multisort_entries(), ldap_result(), ldap_search_s().

ldap_sort_values()

The ldap_sort_values( ) function sorts an array of values retrieved from an ldap_get_values( ) call.

For additional information, see “Sorting the Search Results.”

Syntax

```
#include <ldap.h>
int ldap_sort_values( LDAP *ld, char **vals,
  LDAP_VALCMP_CALLBACK cmp );
```

Parameters

This function has the following parameters:

Table 18-123 ldap_sort_values() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`vals`	The array of values to sort.
`cmp`	Comparison function used when sorting the values. In the `ldap_sort_values()` function, the comparison function must pass `(char **)` parameters. Because of this, you need to use the `ldap_sort_strcasecmp()` function, rather than a function like `strcasecmp()`.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- If unsuccessful, returns an LDAP error code. (See Chapter 19, “Result Codes” for a complete listing of error codes.)

Example

The following section of code sorts the values of attributes before printing them.

Code Example 18-49 ldap_sort_values() code example

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a, *dn;
char **vals;
int i;
char *my_searchbase = "o=airius.com";
char *my_filter = "(sn=Jensen)";
...
```

Code Example 18-49 ldap_sort_values() code example

```
#include <stdio.h>
    if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
      /* Sort the values of the attribute */
      if ( ldap_sort_values( ld, vals, strcasecmp ) ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_sort_values" );
        return( 1 );
      }
      /* Print the values of the attribute */
      for ( i = 0; vals[i] != NULL; i++ ) {
        printf( "%s: %s\n", a, vals[i] );
      }
      /* Free the values from memory */
      ldap_value_free( vals );
    }
...
```

See Also

`ldap_get_values()`, `ldap_sort_strcasecmp()`.

ldap_sort_strcasecmp()

The `ldap_sort_strcasecmp()` routine compares two strings and ignores any differences in case when comparing uppercase and lowercase characters. This function is similar to the C function `strcasecmp()`.

When sorting attribute values with `ldap_sort_values()`, call this function to compare the attribute values.

Syntax

```
#include <ldap.h>
int ldap_sort_strcasecmp( const char **a, const char **b );
```

Parameters

This function has the following parameters:

Table 18-124 ldap_sort_strcasecmp() function parameters

a	Pointer to first string to compare
b	Pointer to second string to compare

Returns

One of the following values:

- If a is greater than b, returns a value greater than 0.
- If a is equal to b, returns 0.
- If a is less than b, returns a value less than 0.

See Also

ldap_sort_values().

ldap_unbind()

The ldap_unbind() function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure.

The ldap_unbind() routine works the same as ldap_unbind_s(); both routines are synchronous. This function is provided so that the function ldap_simple_bind() has a corresponding unbind function. For additional information, see "Closing the Connection to the Server."

Syntax

```
#include <ldap.h>
int ldap_unbind( LDAP *ld );
```

Parameters

This function has the following parameters:

Table 18-125 ldap_unbind() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.

Returns

For a list of possible result codes for an LDAP unbind operation, see the result code documentation for the ldap_unbind_s() function.

Example

The following code closes the current connection with the LDAP server:

Code Example 18-50 ldap_unbind() code example

```
#include <ldap.h>
...
LDAP *ld;
...
/* After completing your LDAP operations with the server, close the
connection. */
if ( ldap_unbind( ld ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "Error while unbinding from the directory" );
  return( 1 );
}
...
```

See Also

`ldap_unbind_s(), ldap_unbind_ext().`

ldap_unbind_s()

The `ldap_unbind_s()` function unbinds from the directory, terminates the current association, and frees the resources contained in the `LDAP` structure.

Syntax

```
#include <ldap.h>
int ldap_unbind_s( LDAP *ld );
```

Parameters

This function has the following parameters:

Table 18-126 ldap_unbind_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request.

- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The three unbind functions (ldap_unbind_ext(), ldap_unbind() and ldap_unbind_s()) all work synchronously in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning.

Note that there is no server response to an LDAP unbind operation. All three of the unbind functions return LDAP_SUCCESS (or another LDAP error code if the request cannot be sent to the LDAP server). After a call to one of the unbind functions, the session handle ld is invalid and it is illegal to make any further LDAP API calls using ld. For additional information, see "Closing the Connection to the Server."

Example

The following code closes the current connection with the LDAP server:

Code Example 18-51 ldap_unbind_s() code example

```
#include <ldap.h>
...
LDAP *ld;
...
/* After completing your LDAP operations with the server, close the
connection. */
if ( ldap_unbind_s( ld ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "Error while unbinding from the directory" );
  return( 1 );
}
...
```

See Also

ldap_unbind(), ldap_unbind_ext().

ldap_unbind_ext()

The ldap_unbind_ext() function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure.

Unlike the other two unbind functions, ldap_unbind_ext() allows you to explicitly include both server and client controls in your unbind request. However, since there is no server response to an unbind request, there is no way to receive a response from a server control that is included with your unbind request.

Syntax

```
#include <ldap.h>
int ldap_unbind_ext( LDAP *ld, LDAPControl **serverctrls,
LDAPControl **clientctrls );
```

Parameters

This function has the following parameters:

Table 18-127 ldap_unbind_ext() function parameters

ld	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3 protocol. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see "Specifying the LDAP Version of Your Client.").

Description

The three unbind functions (ldap_unbind_ext(), ldap_unbind() and ldap_unbind_s()) all work synchronously in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning.

Note that there is no server response to an LDAP unbind operation. All three of the unbind functions return LDAP_SUCCESS (or another LDAP error code if the request cannot be sent to the LDAP server). After a call to one of the unbind functions, the session handle ld is invalid and it is illegal to make any further LDAP API calls using ld. For additional information, see "Closing the Connection to the Server."

See Also

ldap_unbind(), ldap_unbind_s().

ldap_url_parse()

The ldap_url_parse() function parses an LDAP URL into its components. For more information, see "Getting the Components of an LDAP URL."

Syntax

```
#include <ldap.h>
int ldap_url_parse( const char *url, LDAPURLDesc **ludpp );
```

Parameters

This function has the following parameters:

Table 18-128 ldap_url_parse() function parameters

url	The URL that you want to check.
ludpp	Pointer to a structure containing the components of the URL.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_URL_ERR_NOTLDAP if the URL does not begin with the "ldap://" or "ldaps://" prefix.
- LDAP_URL_ERR_NODN if the URL missing trailing slash after host or port.
- LDAP_URL_ERR_BADSCOPE if the scope within the URL is invalid.
- LDAP_URL_ERR_MEM if not enough free memory is available for this operation.
- LDAP_URL_ERR_PARAM if an invalid argument was passed to the function.

Example

The following section of code parses an LDAP URL and prints out each component of the URL.

Code Example 18-52 ldap_url_parse() code example

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url =
"ldap://ldap.netscape.com:5000/o=airius.com?cn,mail,telephoneNumber?sub?(sn=J
ensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
  switch( res ){
    case LDAP_URL_ERR_NOTLDAP:
      printf( "URL does not begin with \"ldap://\"\n" );
      break;
    case LDAP_URL_ERR_NODN:
      printf( "URL missing trailing slash after host or port\n" );
      break;
    case LDAP_URL_ERR_BADSCOPE:
      printf( "URL contains an invalid scope\n" );
      break;
    case LDAP_URL_ERR_MEM:
      printf( "Not enough memory\n" );
      break;
    default:
      printf( "Unknown error\n" );
  }
  return( 1 );
}
printf( "Components of the URL:\n" );
printf( "Host name: %s\n", ludpp->lud_host );
printf( "Port number: %d\n", ludpp->lud_port );
if ( ludpp->lud_dn != NULL ) {
  printf( "Base entry: %s\n", ludpp->lud_dn );
} else {
  printf( "Base entry: Root DN\n" );
}
if ( ludpp->lud_attrs != NULL ) {
  printf( "Attributes returned: \n" );
  for ( i=0; ludpp->lud_attrs[i] != NULL; i++ ) {
    printf( "\t%s\n", ludpp->lud_attrs[i] );
}
} else {
       printf( "No attributes returned.\n" );
}
printf( "Scope of the search: " );
switch( ludpp->lud_scope ) {
  case LDAP_SCOPE_BASE:
    printf( "base\n" );
```

Code Example 18-52 ldap_url_parse() code example

```
#include <stdio.h>
    break;
  case LDAP_SCOPE_ONELEVEL:
    printf( "one\n" );
    break;
  case LDAP_SCOPE_SUBTREE:
    printf( "sub\n" );
    break;
  default:
    printf( "Unknown scope\n" );
}
printf( "Filter: %s\n", ludpp->lud_filter );
...
```

See Also

`ldap_free_urldesc()`.

ldap_url_search()

The `ldap_url_search()` function searches the directory for matching entries, based on the contents of the URL.

`ldap_url_search()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_url_search_s()` instead. For more information on asynchronous and synchronous functions, see “Calling Synchronous and Asynchronous Functions.”

For more information on processing LDAP searches specified as URLs, see “Processing an LDAP URL.”

Syntax

```
#include <ldap.h>
int ldap_url_search( LDAP *ld, const char *url, int attrsonly );
```

Parameters

This function has the following parameters:

Table 18-129 ldap_url_search() function parameters

`ld`	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
`url`	LDAP URL specifying a search of the directory.

Table 18-129 ldap_url_search() function parameters

`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.

Returns

Returns the message ID of the `ldap_url_search()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`.

Example

Code Example 18-53 ldap_url_search() code example

```
#include "examples.h"

static void do_other_work();
unsigned long global_counter = 0;

int
main( int argc, char **argv )
{
  char *my_url =
"ldap://ldap.netscape.com/o=airius.com?cn,mail,telephoneNumber?sub?(sn=Jensen
)";
  LDAP *ld;
  LDAPMessage *result, *e;
  BerElement *ber;
  char *a, *dn;
  char **vals;
  int i, rc, finished, msgid;
  int num_entries = 0;
  struct timeval zerotime;

  zerotime.tv_sec = zerotime.tv_usec = 0L;

/* get a handle to an LDAP connection */
if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}
/* authenticate to the directory as nobody */
if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_simple_bind_s" );
  return( 1 );
}
/* search for all entries with surname of Jensen */
```

Code Example 18-53 ldap_url_search() code example

```
#include "examples.h"
if (( msgid = ldap_url_search( ld, my_url, 0 )) == -1 ) {
  ldap_perror( ld, "ldap_url_search" );
  return( 1 );
}

/* Loop, polling for results until finished */
finished = 0;
while ( !finished ) {
  /*
  * Poll for results.   We call ldap_result with the "all" parameter
  * set to zero. This causes ldap_result() to return exactly one
  * entry if at least one entry is available. This allows us to
  * display the entries as they are received.
  */
  result = NULL;
  rc = ldap_result( ld, msgid, 0, &zerotime, &result );
  switch ( rc ) {
  case -1:
    /* some error occurred */
    ldap_perror( ld, "ldap_result" );
    return( 1 );
  case 0:
  /* Timeout was exceeded. No entries are ready for retrieval. */
    if ( result != NULL ) {
      ldap_msgfree( result );
    }
    break;
  default:
  /*
  * Either an entry is ready for retrieval, or all entries have
  * been retrieved.
  */
    if (( e = ldap_first_entry( ld, result )) == NULL ) {
      /* All done */
      finished = 1;
      if ( result != NULL ) {
        ldap_msgfree( result );
      }
      continue;
      }
    /* for each entry print out name + all attrs and values */
    num_entries++;
    if (( dn = ldap_get_dn( ld, e )) != NULL ) {
      printf( "dn: %s\n", dn );
      ldap_memfree( dn );
    }
    for ( a = ldap_first_attribute( ld, e, &ber );
     a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
      if (( vals = ldap_get_values( ld, e, a )) != NULL ) {
        for ( i = 0; vals[ i ] != NULL; i++ ) {
          printf( "%s: %s\n", a, vals[ i ] );
        }
      ldap_value_free( vals );
```

Code Example 18-53 ldap_url_search() code example

```
#include "examples.h"
      }
      ldap_memfree( a );
    }
    if ( ber != NULL ) {
      ldap_ber_free( ber, 0 );
    }
    printf( "\n" );
    ldap_msgfree( result );
  }
  /* Do other work here while you are waiting... */
  do_other_work();
}

/* All done. Print a summary. */
printf( "%d entries retrieved. I counted to %ld "
      "while I was waiting.\n", num_entries,
      global_counter );
ldap_unbind( ld );
return( 0 );
}

/*
 * Perform other work while polling for results. */
static void
do_other_work()
{
    global_counter++;
}
```

See Also

`ldap_url_search_s(), ldap_result(), ldap_result2error().`

ldap_url_search_s()

The `ldap_url_search_s()` function searches the directory for matching entries, based on the contents of the URL.

`ldap_url_search_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_url_search()` instead. (For more information on asynchronous and synchronous functions, see "Calling Synchronous and Asynchronous Functions.")

For more information on processing LDAP searches specified as URLs, see "Processing an LDAP URL."

Syntax

```
#include <ldap.h>
int ldap_url_search_s( LDAP *ld, const char *url,
  int attrsonly, LDAPMessage **res );
```

Parameters

This function has the following parameters:

Table 18-130 ldap_url_search_s() function parameters

`ld`	Connection handle, which is a pointer to an `LDAP` structure containing information about the connection to the LDAP server.
`url`	LDAP URL specifying a search of the directory.
`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • `0` specifies that both attribute types and attribute values are returned. • `1` specifies that only attribute types are returned.
`res`	Results of the search (when the call is completed).

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- If unsuccessful, returns the LDAP error code for the operation. See Chapter 19, “Result Codes” for a complete listing.

Example

The following example processes a search request from an LDAP URL.

Code Example 18-54 ldap_url_search_s() code example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_url =
"ldap://ldap.netscape.com/o=airius.com?cn,mail,telephoneNumber?sub?(sn=Jensen
)";
...
/* Process the search request in the URL */
```

Code Example 18-54 ldap_url_search_s() code example

```
#include <stdio.h>
if ( ldap_url_search_s( ld, my_url, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_url_search_s" );
  return( 1 );
}
...
```

See Also

`ldap_search()`, `ldap_search_st()`.

ldap_url_search_st()

The `ldap_url_search_st()` function searches the directory for matching entries, based on the contents of the URL. This function works like `ldap_url_search_s()` and lets you specify a timeout period for the search.

For more information, see "Processing an LDAP URL."

Syntax

```
#include <ldap.h>
int ldap_url_search_st( LDAP *ld, const char *url, int attrsonly,
  struct timeval *timeout, LDAPMessage **res );
```

Parameters

This function has the following parameters:

Table 18-131 ldap_url_search_st() function parameters

`ld`	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
`url`	LDAP URL specifying a search of the directory.
`attrsonly`	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • 0 specifies that both attribute types and attribute values are returned. • 1 specifies that only attribute types are returned.
`timeout`	Maximum time to wait for the results of the search.
`res`	Results of the search (when the call is completed).

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns the LDAP error code for the operation. See Chapter 19, “Result Codes” for a complete listing.

See Also

ldap_search(), ldap_search_s().

ldap_value_free()

The ldap_value_free() function frees an array of values from memory. Use the ldap_value_free_len() function instead of this function if the values are berval structures.

For additional information, see “Getting the Values of an Attribute.”

Syntax

```
#include <ldap.h>
void ldap_value_free( char **values );
```

Parameters

This function has the following parameters:

Table 18-132 ldap_value_free() function parameters

values	Array of values.

Example

See the example under ldap_get_values().

See Also

ldap_get_values(), ldap_value_free_len().

ldap_value_free_len()

The ldap_value_free_len() function frees an array of berval structures from memory. Use the ldap_value_free() function instead of this function if the values are string values.

For additional information, see "Getting the Values of an Attribute."

Syntax

```
#include <ldap.h>
void ldap_value_free_len( struct berval **values );
```

Parameters

This function has the following parameters:

Table 18-133 ldap_value_free_len() function parameters

values	Array of berval structures.

Example

See the example under ldap_get_values_len().

See Also

ldap_get_values(), ldap_get_values_len().

ldap_version()

This function has been deprecated; you should use the function ldap_get_option() in its place (it is documented here for backward compatibility only).

Gets version information about the LDAP SDK for C libraries. The version information is returned in an LDAPVersion structure.

Syntax

```
#include <ldap.h>
int ldap_version( LDAPVersion *ver );
```

Parameters

This function has the following parameters:

Table 18-134 ldap_version() function parameters

ver	LDAPVersion structure returning version information. If you only want the SDK version, you can pass NULL for this parameter.

Returns

The version number of the LDAP SDK for C, multiplied by 100. For example, for version 1.0 of the LDAP SDK for C, the function returns 100.

ldapssl_advclientauth_init()

Initializes your client application to connect to a secure LDAP server over SSL and to use certificate-based client authentication.

The use of `ldapssl_pkcs_init()`, a version 4.0 function, is recommended over the use of this older SSL initialization function.

Syntax

```
#include <ldap_ssl.h>
int LDAP_CALL ldapssl_advclientauth_init( char *certdbpath,
  void *certdbhandle, int needkeydb, char *keydbpath,
  void *keydbhandle, int needsecmoddb, char *secmodpath,
  const int sslstrength);
```

Parameters

This function has the following parameters:

Table 18-135 ldapssl_advclientauth_init() function parameters

`certdbpath`	Path to the database containing certificates for your client. The database must be the `cert7.db` certificate database used by Netscape Communicator 4.*x*. Note the following: • You can include the database filename in the path (for example, `/usr/mozilla/cert7.db`). • If you specify the path to the directory containing the certificate database (for example, `/usr/mozilla`), the function assumes that the database file is named `cert7.db`. • If you pass `NULL` for this parameter, the function looks for the certificate database used by Netscape Communicator (for example, `~/.netscape/cert7.db` on UNIX).
`certdbhandle`	Pass a `NULL` value for this. (This parameter is not used currently.)

Table 18-135 ldapssl_advclientauth_init() function parameters

`needkeydb`	Specifies whether or not the private key database needs to be opened for use. This parameter can have one of the following values: • If it is a non-zero value, the function opens the private key database, which is identified by the `keydbpath` argument. • If `0`, the function does not open the private key database.
`keydbpath`	Path to the database containing the private key certified by your certificate. The database must be the `key3.db` private key database used by Netscape Communicator 4.*x*. Note the following: • You can include the database filename in the path (for example, `/usr/mozilla/key3.db`). • If you specify the path to the directory containing the private key database (for example, `/usr/mozilla`), the function assumes that the database file is named `key3.db`. • If you pass `NULL` for this parameter, the function looks for the private key database used by Netscape Communicator (for example, `~/.netscape/key3.db` on UNIX).
`certdbhandle`	Pass a `NULL` value for this. (This parameter is not used currently.)
`needsecmoddb`	Specifies whether or not the security module database file needs to be opened for use. This parameter can have one of the following values: • If it is a non-zero value, the function opens the security module database, which is identified by the `keydbpath` argument. • If `0`, the function does not open the security modules database.

Table 18-135 ldapssl_advclientauth_init() function parameters

`secmodpath`	Path to the database containing security modules. The database must be the `secmod.db` private key database used by Netscape Communicator 4.*x*. Note the following: • You can include the database filename in the path (for example, `/usr/mozilla/secmod.db`). • If you specify the path to the directory containing the security module database (for example, `/usr/mozilla`), the function assumes that the database file is named `secmod.db`. • If you pass `NULL` for this parameter, the function looks for the security module database used by Netscape Communicator (for example, `~/.netscape/secmod.db` on UNIX).
`sslstrength`	Specifies how the server certificate is evaluated. You can specify one of the following values: • `LDAPSSL_AUTH_WEAK` indicates that you accept the server's certificate without checking the CA who issued the certificate. • `LDAPSSL_AUTH_CERT` indicates that you accept the server's certificate only if you trust the CA who issued the certificate. • `LDAPSSL_AUTH_CNCHECK` indicates that you accept the server's certificate only if you trust the CA who issued the certificate and if the value of the cn attribute is the DNS hostname of the server.

Returns

One of the following values:

- `0` if successful.
- `-1` if unsuccessful.

Description

You can call the `ldapssl_advclientauth_init()` function to initialize your client application for SSL and for certificate-based client authentication.

This function is similar to `ldapssl_clientauth_init()` and allows you to do the following:

- Specify the name and path of a security module database.
- Specify the method used to verify the server's certificate.

You must call this function before calling the `ldapssl_init()` function to connect to the server. For details, see Chapter 12, "Connecting Over SSL."

Example

The following example initializes a client before connecting with a secure LDAP server.

Code Example 18-55 ldapssl_advclientauth_init() code example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_advclientauth_init( "/u/mozilla/.netscape/cert7.db",
  NULL, 1, "/u/mozilla/.netscape/key3.db", NULL , 1,
  "/u/mozilla/.netscape/secmod.db", LDAPSSL_AUTH_CNCHECK) < 0 ) {
  perror( "ldap_advclientauth_init" );
    return( 1 );
  }
}
...
```

See Also

`ldap_init()`, `ldapssl_clientauth_init()`, `ldapssl_init()`, `ldapssl_install_routines()`.

ldapssl_client_init()

Initializes your client application to connect to a secure LDAP server over SSL.

The use of `ldapssl_pkcs_init()`, a version 4.0 function, is recommended over the use of this older SSL initialization function.

Syntax

```
#include <ldap_ssl.h>
int ldapssl_client_init( const char *certdbpath, void *certdbhandle
);
```

Parameters

This function has the following parameters:

Table 18-136 ldapssl_client_init() function parameters

`certdbpath`	Path to the database containing certificates for your client. The database must be the `cert7.db` certificate database used by Netscape Communicator 4.*x*. You can either specify the path to the directory containing the certificate database (in which case the function assumes that the database file is named `cert7.db`) or you can include the database filename in the path. If you pass `NULL` for this parameter, the function looks for the certificate database used by Netscape Communicator (for example, `~/.netscape/cert7.db` on UNIX).
`certdbhandle`	Pass a `NULL` value for this. (This parameter is not used currently.)

Returns

One of the following values:

- `0` if successful.
- `-1` if unsuccessful.

Description

You can call the `ldapssl_client_init()` function to initialize your client application for SSL.

If you plan to use certificate-based authentication, you should call the `ldapssl_clientauth_init()` or `ldapssl_advclientauth_init()` function instead of this function.

You must call this function before calling the `ldapssl_init()` function to connect to the server. For details, see Chapter 12, “Connecting Over SSL.”

Example

The following example initializes a client before connecting with a secure LDAP server.

Code Example 18-56 ldapssl_client_init() code example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert7.db", NULL ) < 0) {
  printf( "Failed to initialize SSL client...\n" );
  return( 1 );
}
...
```

See Also

`ldap_init()`, `ldapssl_init()`, `ldapssl_install_routines()`.

ldapssl_clientauth_init()

Initializes your client application to connect to a secure LDAP server over SSL and to use certificate-based client authentication.

The use of `ldapssl_pkcs_init()`, a version 4.0 function, is recommended over the use of this older SSL initialization function.

Syntax

```
#include <ldap_ssl.h>
int ldapssl_clientauth_init( char *certdbpath, void *certdbhandle,
  int needkeydb, char *keydbpath, void *keydbhandle );
```

Parameters

This function has the following parameters:

Table 18-137 ldapssl_clientauth_init() function parameters

`certdbpath`	Path to the database containing certificates for your client. The database must be the `cert7.db` certificate database used by Netscape Communicator 4.*x*. You can either specify the path to the directory containing the certificate database (in which case the function assumes that the database file is named `cert7.db`) or you can include the database filename in the path. If you pass `NULL` for this parameter, the function looks for the certificate database used by Netscape Communicator (for example, `~/.netscape/cert7.db` on UNIX).
`certdbhandle`	Pass a `NULL` value for this. (This parameter is not used currently.)
`needkeydb`	Specifies whether or not the private key database needs to be opened for use. This parameter can have one of the following values: • If it is a non-zero value, the function opens the private key database, which is identified by the `keydbpath` argument. • If `0`, the function does not open the private key database.
`keydbpath`	Path to the database containing the private key certified by your certificate. The database must either be the `key3.db` certificate database used by Netscape Communicator 4.*x*. You can either specify the path to the directory containing the private key database (in which case the function assumes that the database file is named `key3.db`) or you can include the database filename in the path. If you pass NULL for this parameter, the function looks for the key database used by Netscape Communicator (for example, `~/.netscape/key3.db` on UNIX).
`certdbhandle`	Pass a `NULL` value for this. (This parameter is not used currently.)

Returns

One of the following values:

- `0` if successful.
- `-1` if unsuccessful.

Description

You can call the `ldapssl_clientauth_init()` function to initialize your client application for SSL and certificate-based client authentication.

If you need to specify the name and path of the security modules database or if you need to specify how the server's certificate will be verified, you should call the `ldapssl_advclientauth_init()` function instead of this function.

You must call this function before calling the `ldapssl_init()` function to connect to the server. For details, see Chapter 12, "Connecting Over SSL."

Example

The following example initializes a client before connecting with a secure LDAP server.

Code Example 18-57 ldapssl_clientauth_init() code example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_clientauth_init( "/u/mozilla/.netscape/cert7.db", NULL, 1,
  "/u/mozilla/.netscape/key3.db", NULL ) < 0 ) {
  perror( "ldap_clientauth_init" );
    return( 1 );
  }
}
...
```

See Also

`ldap_init()`, `ldapssl_clientauth_init()`, `ldapssl_init()`, `ldapssl_install_routines()`.

ldapssl_enable_clientauth()

Enables your application to use client authentication.

Syntax

```
#include <ldap_ssl.h>
int ldapssl_enable_clientauth( LDAP *ld, char *keynickname,
  char *keypasswd, char *certnickname );
```

Parameters

This function has the following parameters:

Table 18-138 ldapssl_enable_clientauth() function parameters

`ld`	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
`keynickname`	This parameter is not currently used. Pass an empty string ("") for this value.
`keypasswd`	Password to the encrypted private key database.
`certnickname`	Nickname of the certificate that you want to use for client authentication.

Returns

One of the following values:

- `0` if successful.
- `-1` if unsuccessful.

See Also

`ldapssl_clientauth_init()`.

ldapssl_err2string()

The `ldapssl_err2string()` function returns the corresponding error message for an SSL-specific error code. For more information, see “Getting the Error Message.”

Syntax

```
#include <ldap_ssl.h>
const char * LDAP_CALL ldapssl_err2string ( const int prerrno );
```

Parameters

This function has the following parameters:

Table 18-139 ldapssl_err2string() function parameters

`prerrno`	The SSL error code that you want interpreted into an error message.

Returns

One of the following values:

- If successful, returns the corresponding error message for the SSL error code.
- If unsuccessful (for example, if the error code is not a known SSL error code), returns a pointer to the string "`Unknown error`."

Description

This function provides support for the SSL-specific error messages that are not covered by the regular message routine `ldap_err2string()`. If any `ldapssl_*()` function returns an error code that is unknown to `ldap_err2string()` (and `ldap_err2string()` returns "`Unknown Error`"), this function should be called to determine the SSL-specific error message.

To check for SSL errors, you should call `ldapssl_err2string()` after you call any of the following SSL initialization functions:

- `ldapssl_client_init()`
- `ldapssl_clientauth_init()`
- `ldapssl_advclientauth_init()`
- `ldapssl_pkcs_init()`

The errors returned by these functions are usually related to certificate database corruption, missing certs in a certificate database, client authentication failures, and other general SSL errors.

See Also

`ldapssl_client_init()`, `ldapssl_clientauth_init()`, `ldapssl_advclientauth_init()`, `ldapssl_pkcs_init()`, `ldap_err2string()`

ldapssl_init()

Initializes the LDAP library for SSL and installs the I/O routines for SSL. `ldapssl_init()` allocates an LDAP structure but does not open an initial connection.

Calling this function is equivalent to calling `ldap_init()` followed by `ldapssl_install_routines()` and `ldap_set_option()` to set the `LDAP_OPT_SSL` option to `LDAP_OPT_ON`.

Before calling this function, you should call the `ldapssl_client_init()` function to initialize your client for SSL. For details, see Chapter 12, "Connecting Over SSL."

Syntax

```
#include <ldap_ssl.h>
LDAP *ldapssl_init( const char *defhost, int defport,
  int defsecure );
```

Parameters

This function has the following parameters:

Table 18-140 ldapssl_init() function parameters

`defhost`	Connect to this LDAP server, if no other server is specified.
`defport`	Connect to this server port, if no other port is specified. To specify the default port 389, use `LDAP_PORT` as the value for this parameter.
`defsecure`	Determines whether or not to establish the default connection over SSL. Set this to a non-zero value to establish the default connection over SSL.

Returns

One of the following values:

- If successful, returns a pointer to an `LDAP` structure, which should be passed to subsequent calls to other LDAP API functions.
- If unsuccessful, returns a `-1`.

Example

The following example connects your client with a secure LDAP server.

Code Example 18-58 ldapssl_init() code example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert5.db", NULL ) < 0) {
  printf( "Failed to initialize SSL client...\n" );
  return( 1 );
}
/* get a handle to an LDAP connection */
if ( (ld = ldapssl_init( "cert.netscape.com", LDAPS_PORT, 1 )) == NULL {
  perror( "ldapssl_init" );
  return( 1 );
}
```

Code Example 18-58 ldapssl_init() code example

```
#include <ldap.h>
...
/* Client can now perform LDAP operations on the secure LDAP server */
...
```

See Also

`ldap_init()`, `ldapssl_client_init()`, `ldapssl_install_routines()`.

ldapssl_install_routines()

Installs the I/O routines to enable SSL over LDAP. You need to call this routine in combination with `ldap_init()` and `ldap_set_option()`. (As an alternative, you can call `ldapssl_init()` instead of these three functions).

As is the case with the `ldapssl_init()` function, you need to call `ldapssl_client_init()` before calling this function.

Syntax

```
#include <ldap_ssl.h>
int ldapssl_install_routines( LDAP *ld );
```

Parameters

This function has the following parameters:

Table 18-141 ldapssl_install_routines() function parameters

`ld`	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.

Returns

One of the following values:

- `0` if successful.
- `-1` if unsuccessful.

Example

The following example connects your client with a secure LDAP server.

Code Example 18-59 ldapssl_install_routines() code example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert5.db", NULL ) < 0) {
  printf( "Failed to initialize SSL client...\n" );
  return( 1 );
}
/* Get the handle to an LDAP connection */
if ( (ld = ldap_init( MY_HOST, 6360 )) == NULL ) {
  perror( "ldap_init" );
  return( 1 );
}

/* Load SSL routines */
if ( ldapssl_install_routines( ld ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldapssl_install_routines" );
  return( 1 );
}

/* Set the option to use SSL with the default connection */
if (ldap_set_option( ld, LDAP_OPT_SSL, LDAP_OPT_ON ) != LDAP_SUCCESS) {
  ldap_perror( ld, "ldap_set_option" );
  return( 1 );
}
...
```

See Also

`ldap_init()`, `ldapssl_init()`, `ldapssl_client_init()`.

ldapssl_pkcs_init()

This function, added in version 4.0 of the LDAP C SDK, provides better SSL initialization than the previous `ldapssl_*_init()` SSL initialization functions `ldapssl_client_init()`, `ldapssl_clientauth_init()`, and `ldapssl_advclientauth_init()`.

The function `ldap_pcks_init()` is preferred over these previous initialization because it is thread-safe, while the other `ldapssl_*_init()` functions are not.

The LDAP SDK for C uses the Public Key Cryptography Standard (PKCS) API implemented in Network Security Services (NSS) to provide SSL security support. Specifically, NSS implements the security API as defined in the PKCS#11 standard.

Syntax

```
#include <ldap_ssl.h>
int LDAP_CALL ldapssl_pkcs_init(const struct
  ldapssl_pkcs_fns *pfns);
```

Parameters

This function has the following parameters:

Table 18-142 ldapssl_pkcs_init() function parameters

`pfns`	A pointer to a `ldapssl_okcs_fns` structure that defines the callback function for obtaining the required SSL security parameters.

Returns

One of the following values:

- `0` is successful.
- `-1` if unsuccessful.
- *n*, a positive integer, denotes an NSPR error as returned by the `PR_GetError()` NSPR function. Consult the NSPR documentation on `mozilla.org` for more information.

Description

The `ldap_pkcs_init() structure` sets up callbacks for the security library to obtain required runtime information. It should be used in place of `ldapssl_client_init()`, `ldapssl_clientauth_init()`, and `ldapssl_advclientauth_init()`.

Because `ldap_pkcs_init()` is based on the `ldapssl_pkcs_fns` structure, you do not need to know all of the security parameters at initialization, unlike the other SSL initialization functions (`ldapssl_*_init()`), which did require all security parameters to be known at the time of initialization.

Over time, the `ldapssl_*_int()` functions will be deprecated in favor of `ldap_pcks_init()`.

The `ldapssl_pkcs_fns` structure is defined as follows:

```
typedef int (LDAP_PKCS_GET_TOKEN_CALLBACK)
  (void *context, char **tokenname);
typedef int (LDAP_PKCS_GET_PIN_CALLBACK)
  (void *context, const char *tokenname, char **tokenpin);
typedef int (LDAP_PKCS_GET_CERTPATH_CALLBACK)
  (void *context, char **certpath);
```

```
typedef int (LDAP_PKCS_GET_KEYPATH_CALLBACK)(void *context,
  char **keypath);
typedef int (LDAP_PKCS_GET_MODPATH_CALLBACK)
  (void *context, char **modulepath);
typedef int (LDAP_PKCS_GET_CERTNAME_CALLBACK)
  (void *context, char **certname);
typedef int (LDAP_PKCS_GET_DONGLEFILENAME_CALLBACK)
  (void *context, char **filename);

#define PKCS_STRUCTURE_ID 1
struct ldapssl_pkcs_fns {
  int local_structure_id;
  void *local_data;
  LDAP_PKCS_GET_CERTPATH_CALLBACK *pkcs_getcertpath;
  LDAP_PKCS_GET_CERTNAME_CALLBACK *pkcs_getcertname;
  LDAP_PKCS_GET_KEYPATH_CALLBACK *pkcs_getkeypath;
  LDAP_PKCS_GET_MODPATH_CALLBACK *pkcs_getmodpath;
  LDAP_PKCS_GET_PIN_CALLBACK *pkcs_getpin;
  LDAP_PKCS_GET_TOKEN_CALLBACK *pkcs_gettokenname;
  LDAP_PKCS_GET_DONGLEFILENAME_CALLBACK *pkcs_getdonglefilename;
};
```

See Also

`ldapssl_client_init()`, `ldapssl_clientauth_init()`, `ldapssl_advclientauth_init()`.

Chapter 19

Result Codes

This chapter lists the result codes that can be returned by functions in the LDAP SDK for C. These codes are specified in the LDAP protocol.

For ease of reference, this chapter lists the result codes in two formats:

- Result Codes Listed Alphabetically
- Result Codes Listed in Numerical Order

Result Codes Listed Alphabetically

- `LDAP_ADMINLIMIT_EXCEEDED`
- `LDAP_AFFECTS_MULTIPLE_DSAS`
- `LDAP_ALIAS_DEREF_PROBLEM`
- `LDAP_ALIAS_PROBLEM`
- `LDAP_ALREADY_EXISTS`
- `LDAP_AUTH_UNKNOWN`
- `LDAP_BUSY`
- `LDAP_CLIENT_LOOP`
- `LDAP_COMPARE_FALSE`
- `LDAP_COMPARE_TRUE`
- `LDAP_CONFIDENTIALITY_REQUIRED`
- `LDAP_CONNECT_ERROR`
- `LDAP_CONSTRAINT_VIOLATION`

- LDAP_CONTROL_NOT_FOUND
- LDAP_DECODING_ERROR
- LDAP_ENCODING_ERROR
- LDAP_FILTER_ERROR
- LDAP_INAPPROPRIATE_AUTH
- LDAP_INAPPROPRIATE_MATCHING
- LDAP_INDEX_RANGE_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_INVALID_CREDENTIALS
- LDAP_INVALID_DN_SYNTAX
- LDAP_INVALID_SYNTAX
- LDAP_IS_LEAF
- LDAP_LOCAL_ERROR
- LDAP_LOOP_DETECT
- LDAP_MORE_RESULTS_TO_RETURN
- LDAP_NAMING_VIOLATION
- LDAP_NO_MEMORY
- LDAP_NO_OBJECT_CLASS_MODS
- LDAP_NO_RESULTS_RETURNED
- LDAP_NO_SUCH_ATTRIBUTE
- LDAP_NO_SUCH_OBJECT
- LDAP_NOT_ALLOWED_ON_NONLEAF
- LDAP_NOT_ALLOWED_ON_RDN
- LDAP_NOT_SUPPORTED
- LDAP_OBJECT_CLASS_VIOLATION
- LDAP_OPERATIONS_ERROR
- LDAP_OTHER

- LDAP_PARAM_ERROR
- LDAP_PARTIAL_RESULTS
- LDAP_PROTOCOL_ERROR
- LDAP_REFERRAL
- LDAP_REFERRAL_LIMIT_EXCEEDED
- LDAP_RESULTS_TOO_LARGE
- LDAP_SASL_BIND_IN_PROGRESS
- LDAP_SERVER_DOWN
- LDAP_SIZELIMIT_EXCEEDED
- LDAP_SORT_CONTROL_MISSING
- LDAP_STRONG_AUTH_NOT_SUPPORTED
- LDAP_STRONG_AUTH_REQUIRED
- LDAP_SUCCESS
- LDAP_TIMELIMIT_EXCEEDED
- LDAP_TIMEOUT
- LDAP_TYPE_OR_VALUE_EXISTS
- LDAP_UNAVAILABLE
- LDAP_UNAVAILABLE_CRITICAL_EXTENSION
- LDAP_UNDEFINED_TYPE
- LDAP_UNWILLING_TO_PERFORM
- LDAP_USER_CANCELLED

Result Codes Listed in Numerical Order

- 0 - LDAP_SUCCESS
- 1- LDAP_OPERATIONS_ERROR
- 2 - LDAP_PROTOCOL_ERROR
- 3 - LDAP_TIMELIMIT_EXCEEDED

- 4 - LDAP_SIZELIMIT_EXCEEDED
- 5 - LDAP_COMPARE_FALSE
- 6 - LDAP_COMPARE_TRUE
- 7 - LDAP_STRONG_AUTH_NOT_SUPPORTED
- 8 - LDAP_STRONG_AUTH_REQUIRED
- 9 - LDAP_PARTIAL_RESULTS
- 10 - LDAP_REFERRAL
- 11 - LDAP_ADMINLIMIT_EXCEEDED
- 12 - LDAP_UNAVAILABLE_CRITICAL_EXTENSION
- 13 - LDAP_CONFIDENTIALITY_REQUIRED
- 14 - LDAP_SASL_BIND_IN_PROGRESS
- 15 - LDAP_NO_SUCH_ATTRIBUTE
- 17 - LDAP_UNDEFINED_TYPE
- 18 - LDAP_INAPPROPRIATE_MATCHING
- 19 - LDAP_CONSTRAINT_VIOLATION
- 20 - LDAP_TYPE_OR_VALUE_EXISTS
- 21 - LDAP_INVALID_SYNTAX
- 32 - LDAP_NO_SUCH_OBJECT
- 33 - LDAP_ALIAS_PROBLEM
- 324 - LDAP_INVALID_DN_SYNTAX
- 35 - LDAP_IS_LEAF
- 36 - LDAP_ALIAS_DEREF_PROBLEM
- 48 - LDAP_INAPPROPRIATE_AUTH
- 49 - LDAP_INVALID_CREDENTIALS
- 50 - LDAP_INSUFFICIENT_ACCESS
- 51 - LDAP_BUSY
- 52 - LDAP_UNAVAILABLE

- 53- LDAP_UNWILLING_TO_PERFORM
- 54 - LDAP_LOOP_DETECT
- 60 - LDAP_SORT_CONTROL_MISSING
- 61 - LDAP_INDEX_RANGE_ERROR
- 64 - LDAP_NAMING_VIOLATION
- 65 - LDAP_OBJECT_CLASS_VIOLATION
- 66 - LDAP_NOT_ALLOWED_ON_NONLEAF
- 67 - LDAP_NOT_ALLOWED_ON_RDN
- 68 - LDAP_ALREADY_EXISTS
- 69 - LDAP_NO_OBJECT_CLASS_MODS
- 70 - LDAP_RESULTS_TOO_LARGE
- 71 - LDAP_AFFECTS_MULTIPLE_DSAS
- 80- LDAP_OTHER
- 81 - LDAP_SERVER_DOWN
- 82 - LDAP_LOCAL_ERROR
- 83 - LDAP_ENCODING_ERROR
- 84 - LDAP_DECODING_ERROR
- 85 - LDAP_TIMEOUT
- 86 - LDAP_AUTH_UNKNOWN
- 87 - LDAP_FILTER_ERROR
- 88 - LDAP_USER_CANCELLED
- 89 - LDAP_PARAM_ERROR
- 90 - LDAP_NO_MEMORY
- 91 - LDAP_CONNECT_ERROR
- 92 - LDAP_NOT_SUPPORTED
- 93 - LDAP_CONTROL_NOT_FOUND
- 94 - LDAP_NO_RESULTS_RETURNED

- 95 - LDAP_MORE_RESULTS_TO_RETURN
- 96 - LDAP_CLIENT_LOOP
- 97 - LDAP_REFERRAL_LIMIT_EXCEEDED

LDAP_ADMINLIMIT_EXCEEDED

This result code indicates that the "look through limit" on a search operation has been exceeded.

When working with the Directory Server, keep in mind the following:

- If you are bound as the root DN, the server sets an infinite "look through limit".
- If you are not bound as the root DN, the server sets the time limit to the value specified by the `lookthroughtimelimit` directive in the server's `slapd.conf` configuration file.

The "look through limit" is the maximum number of entries that the server will check when gathering a list of potential search result candidates. See the *Directory Server Administrator's Guide* for details.

LDAP_AFFECTS_MULTIPLE_DSAS

This result code indicates that the requested operation needs to be performed on multiple servers, where this operation is not permitted.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_ALIAS_DEREF_PROBLEM

This result code indicates that a problem occurred when dereferencing an alias.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_ALIAS_PROBLEM

This result code indicates that a problem occurred when dereferencing an alias.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_ALREADY_EXISTS

This result code indicates that the request is attempting to add an entry that already exists in the directory.

The Directory Server sends this result code back to the client in the following situations:

- The request is an add request, and the entry already exists in the directory.
- The request is a modify DN request, and the new DN of the entry already identifies another entry.
- The request is adding an attribute to the schema, and an attribute with the specified name or OID already exists.

LDAP_AUTH_UNKNOWN

This result code indicates that an unknown authentication method was specified.

The LDAP API library sets this result code if `ldap_bind()` or `ldap_bind_s()` are called and an authentication method other than `LDAP_AUTH_SIMPLE` is specified. (These functions only allow you to use simple authentication.)

LDAP_BUSY

This result code indicates that the server is currently too busy to perform the requested operation.

At this point in time, neither the LDAP API library nor the Directory Server return this result code.

LDAP_CLIENT_LOOP

This result code indicates that the LDAP client (API library) detected a loop, for example, when following referrals.

LDAP_COMPARE_FALSE

This result code is returned after an LDAP compare operation is completed. The result indicates that the specified attribute value is not present in the specified entry.

LDAP_COMPARE_TRUE

This result code is returned after an LDAP compare operation is completed. The result indicates that the specified attribute value is present in the specified entry.

LDAP_CONFIDENTIALITY_REQUIRED

This result code indicates that confidentiality is required for the operation.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_CONNECT_ERROR

This result code indicates that the LDAP client cannot establish a connection (or has lost the connection) with the LDAP server.

The LDAP API library sets this result code. If you have not established an initial connection with the server, verify that you have specified the correct hostname and port number and that the server is running.

If you have lost the connection to the server, see "Handling Failover," on page 89 for instructions on reconnecting to the server.

LDAP_CONSTRAINT_VIOLATION

This result code indicates that a value in the request does not comply with certain constraints.

The Directory Server sends this result code back to the client in the following situations:

- The request adds or modifies the `userpassword` attribute, and one of the following is true:
  - The server is configured to check the password syntax, and the length of the new password is less than the minimum password length.
  - The server is configured to check the password syntax, and the new password is the same as one of the values of the `uid`, `cn`, `sn`, `givenname`, `ou`, or `mail` attributes.
  - The server is configured to keep a history of previous passwords, and the new password is the same as one of the previous passwords.

- The request is a bind request, and the user is locked out of the account. (For example, the server can be configured to lock a user out of the account after a given number of failed attempts to bind to the server.)

LDAP_CONTROL_NOT_FOUND

This result code indicates that a requested LDAP control was not found.

The LDAP API library sets this result code when parsing a server response for controls and not finding the requested controls. For example:

- `ldap_parse_entrychange_control()` is called, but no entry change notification control is found in the server's response.
- `ldap_parse_sort_control()` is called, but no server-side sorting control is found in the server's response.
- `ldap_parse_virtuallist_control()` is called, but no virtual list view response control is found in the server's response.

For more information on controls, see Chapter 14, "Working with LDAP Controls."

LDAP_DECODING_ERROR

This result code indicates that the LDAP client encountered an error when decoding the LDAP response received from the server.

LDAP_ENCODING_ERROR

This result code indicates that the LDAP client encountered an error when encoding the LDAP request to be sent to the server.

LDAP_FILTER_ERROR

This result code indicates that an error occurred when specifying the search filter.

The LDAP API library sets this result code if it cannot encode the specified search filter in an LDAP search request.

LDAP_INAPPROPRIATE_AUTH

This result code indicates that the type of credentials are not appropriate for the method of authentication used.

The Directory Server sends this result code back to the client if simple authentication is used in a bind request, but the entry has no `userpassword` attribute. And if SASL EXTERNAL is attempted on a non-SSL connection.

LDAP_INAPPROPRIATE_MATCHING

This result code indicates that an extensible match filter in a search request contained a matching rule that does not apply to the specified attribute type.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_INDEX_RANGE_ERROR

This result code indicates that the search results exceeded the range specified by the requested offsets.

This result code applies to search requests that contain "virtual list view" controls. For more information on this control, see “Using the Virtual List View Control.”

Note that versions of the Directory Server prior to 4.0 do not support the "virtual list view" control.

LDAP_INSUFFICIENT_ACCESS

This result code indicates that the client has insufficient access to perform the operation.

Check the user that you are authenticating as and the access control lists for the server. For information on checking ACLs on the server, see the *iPlanet Directory Server Administrator’s Guide*.

LDAP_INVALID_CREDENTIALS

This result code indicates that the credentials provided in the request are invalid.

The Directory Server sends this result code back to the client if a bind request contains the incorrect credentials for a user or if a user's password has already expired.

LDAP_INVALID_DN_SYNTAX

This result code indicates than an invalid DN has been specified.

The Directory Server sends this result code back to the client if an add request or a modify DN request specifies an invalid DN. It also sends this code when an SASL_EXTERNAL bind is attempted but certification to DN mapping fails.

LDAP_INVALID_SYNTAX

This result code indicates that the request contains invalid syntax.

The Directory Server sends this result code back to the client in the following situations:

- The server encounters an ACL with invalid syntax.
- The request attempts to add or modify an `aci` attribute, and the value of the attribute is an ACI with invalid syntax.
- The request is a search request with a substring filter, and the syntax of the filter is invalid.
- The request is a modify request that is attempting to modify the schema, but no values are provided (for example, the request might be attempting to delete all values of the `objectclass` attribute).

LDAP_IS_LEAF

This result code indicates that the specified entry is a leaf entry.

Currently, Directory Server does not send this result code back to LDAP clients.

LDAP_LOCAL_ERROR

This result code indicates that an error occurred in the LDAP client.

LDAP_LOOP_DETECT

This result code indicates that the server was unable to perform the requested operation because of an internal loop.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_MORE_RESULTS_TO_RETURN

This result code indicates that there are more results in the chain of results.

The LDAP API library sets this result code when the `ldap_parse_sasl_bind_result()` function is called to retrieve the result code of an operation, and additional result codes from the server are available in the LDAP structure.

LDAP_NAMING_VIOLATION

This result code indicates that the request violates the structure of the DIT.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_NO_MEMORY

This result code indicates that no memory is available.

The LDAP API library sets this result code if a function cannot allocate memory (for example, when creating an LDAP request or an LDAP control).

LDAP_NO_OBJECT_CLASS_MODS

This result code indicates that the request is attempting to modify the object class that should not be modified (for example, a structural object class).

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_NO_RESULTS_RETURNED

This result code indicates that no results were returned from the server.

The LDAP API library sets this result code when the `ldap_parse_result()` function is called but no result code is included in the server's response.

LDAP_NO_SUCH_ATTRIBUTE

This result code indicates that the specified attribute does not exist in the entry.

The Directory Server might send this result code back to the client if, for example, a modify request specifies the modification or removal of a non-existent attribute or if a compare request specifies a non-existent attribute.

LDAP_NO_SUCH_OBJECT

This result code indicates that the server cannot find an entry specified in the request.

The Directory Server sends this result code back to the client if it cannot find a requested entry and if it cannot refer your client to another LDAP server.

LDAP_NOT_ALLOWED_ON_NONLEAF

This result code indicates that the requested operation is allowed only on entries that do not have child entries (entries that are "leaf" entries, as opposed to "branch" entries).

The Directory Server sends this result code back to the client if the request is a delete request or a modify DN request and the entry is a parent entry. (You cannot delete or move a branch of entries in a single operation.)

LDAP_NOT_ALLOWED_ON_RDN

This result code indicates that the requested operation will affect the RDN of the entry.

The Directory Server sends this result code back to the client if the request is a modify request and the request deletes attribute values from the entry that are used in the RDN of the entry. (For example, if the DN is "`uid=bjensen,ou=People,o=airius.com`", the request removes the attribute value "`uid=bjensen`" from the entry.)

LDAP_NOT_SUPPORTED

This result code indicates that the LDAP client is attempting to use functionality that is not supported.

The LDAP API library sets this result code if the client identifies itself as an LDAPv2 client, and the client is attempting to use functionality available in LDAPv3. For example:

- You are passing LDAP controls to a function.
- You are calling `ldap_extended_operation()`, `ldap_extended_operation_s()`, or `ldap_parse_extended_result()` to request an extended operation or to parse an extended response.
- You are calling `ldap_rename()` or `ldap_rename_s()`, and you are specifying a new "superior DN" as an argument.
- You are calling `ldap_sasl_bind()`, `ldap_sasl_bind_s()`, or `ldap_parse_sasl_bind_result()` to request SASL authentication or to parse a SASL bind response.
- You are calling `ldap_parse_virtuallist_control()` to parse a virtual list control from the server's response.

If you want to use these features, make sure to specify that your LDAP client is an LDAPv3 client. For instructions, see "Specifying the LDAP Version of Your Client."

LDAP_OBJECT_CLASS_VIOLATION

This result code indicates that the request specifies a change to an entry or a new entry that does not comply with the server's schema.

The Directory Server sends this result code back to the client in the following situations:

- The request is an add request, and the new entry does not comply with the schema. For example, the new entry does not have all the required attributes, or the entry has attributes that are not allowed in the entry.
- The request is a modify request, and the change will make the entry noncompliant with the schema. For example, the change removes a required attribute or adds an attribute that is not allowed.

Check the server error logs for more information, and check the schema for the type of entry that you are adding or modifying. For more information on the server's schema, see the *iPlanet Directory Server Administrator's Guide*.

LDAP_OPERATIONS_ERROR

This is a general result code indicating that an error has occurred.

The Directory Server might send this code if, for example, memory cannot be allocated on the server.

To troubleshoot this type of error, check the server's error logs. You may need to increase the log level of the server to get additional information. (See the *iPlanet Directory Server Administrator's Guide* for information on setting the log level of the server.)

LDAP_OTHER

This result code indicates than an unknown error has occurred.

At this point in time, neither the LDAP API library nor the Directory Server return this result code.

LDAP_PARAM_ERROR

This result code indicates that an invalid parameter was specified.

The LDAP API library sets this result code if a function was called and invalid parameters were specified (for example, if the LDAP structure is `NULL`).

LDAP_PARTIAL_RESULTS

The Directory Server sends this result code to LDAPv2 clients to refer them to another LDAP server. When sending this code to a client, the server includes a newline-delimited list of LDAP URLs that identify another LDAP server.

If the client identifies itself as an LDAPv3 client in the request, the Directory Server sends an `LDAP_REFERRAL` result code instead of this result code.

LDAP_PROTOCOL_ERROR

This result code indicates that the LDAP client's request does not comply with the LDAP protocol.

The Directory Server sends this result code back to the client in the following situations:

- The server cannot parse the incoming request.
- The request specifies an attribute type that uses a syntax not supported by the server.
- The request is a SASL bind request, but your client identifies itself as an LDAPv2 client.

  Make sure to specify that your LDAP client is an LDAPv3 client. For instructions, see "Specifying the LDAP Version of Your Client."
- The request is a bind request that specifies an unsupported version of the LDAP protocol.

  Make sure to specify that your LDAP client is either an LDAPv2 client or an LDAPv3 client. For instructions, see "Specifying the LDAP Version of Your Client."
- The request is an add or a modify request that specifies the addition of an attribute type to an entry, but no values are specified.
- The request is a modify request, and one of the following is true:
  - An unknown modify operation is specified (an operation other than `LDAP_MOD_ADD`, `LDAP_MOD_DELETE`, and `LDAP_MOD_REPLACE`).
  - No modifications are specified.
- The request is a modify DN request, and one of the following is true:
  - The new RDN is not a valid RDN.
  - A new superior DN is specified, but your client identifies itself as an LDAPv2 client.

  Note that the Directory Server 3.x and 4.0 do not support the ability to move an entry from under one DN to another DN. If you specify a new superior DN, the request will not be processed.
- The request is a search request, and one of the following is true:
  - An unknown scope is specified (a scope other than `LDAP_SCOPE_BASE`, `LDAP_SCOPE_ONELEVEL`, and `LDAP_SCOPE_SUBTREE`).
  - An unknown filter type is specified.
  - The filter type `LDAP_FILTER_GE` or `LDAP_FILTER_LE` is specified, but the type of attribute contains values that cannot be ordered. (For example, if the attribute type uses a binary syntax, the values of the attribute contain binary data, which cannot be sorted.)

- The request contains an extensible filter (a filter using matching rules), but your client identifies itself as an LDAPv2 client. (Make sure to specify that your LDAP client is an LDAPv3 client. For instructions, see “Specifying the LDAP Version of Your Client.”)
- The request contains an extensible filter (a filter using matching rules), but the matching rule is not supported by the server.

- The request is a search request with a server-side sorting control, and one of the following is true:
  - The server does not have a syntax plug-in that supports the attribute used for sorting.
  - The syntax plug-in does not have a function for comparing values of the attribute. (This compare function is used for sorting.)
  - The type of attribute specified for sorting contains values that cannot be sorted in any order. For example, if the attribute type uses a binary syntax, the values of the attribute contain binary data, which cannot be sorted.
  - The server encounters an error when creating the sorting response control (the control to be sent back to the client).
  - When sorting the results, the time limit or the "look through limit" is exceeded. The "look through limit" is the maximum number of entries that the server will check when gathering a list of potential search result candidates. See the *iPlanet Directory Server Administrator’s Guide* for details.
- The request is an extended operation request, and the server does not support that extended operation.

  In the Directory Server, extended operations are supported through extended operation server plug-ins. Make sure that the server is loading a plug-in that supports the extended operation. Check the OID of the extended operation in your LDAP client to make sure that it matches the OID of the extended operation registered in the server plug-in.

  For more information on extended operation server plug-ins, see the *iPlanet Directory Server Plug-In Programmer’s Guide.*
- An authentication method other than `LDAP_AUTH_SIMPLE` or `LDAP_AUTH_SASL` is specified.

To troubleshoot this type of error, check the server's error logs. You may need to increase the log level of the server to get additional information. See the *iPlanet Directory Server Administrator's Guide* for information on setting the log level of the server.

LDAP_REFERRAL

This result code indicates that the server is referring the client to another LDAP server. When sending this code to a client, the server includes a list of LDAP URLs that identify another LDAP server.

This result code is part of the LDAPv3 protocol. For LDAPv2 clients, the Directory Server sends an `LDAP_PARTIAL_RESULTS` result code instead.

LDAP_REFERRAL_LIMIT_EXCEEDED

This result code indicates that the "referral hop limit" was exceeded.

The LDAP API library sets this result code when following referrals, if the client is referred to other servers more times than allowed by the "referral hop limit". For more information about the "referral hop limit", see “Limiting Referral Hops.”

LDAP_RESULTS_TOO_LARGE

This result code indicates that the results of the request are too large.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_SASL_BIND_IN_PROGRESS

This result code is used in multi-stage SASL bind operations. The server sends this result code back to the client to indicate that the authentication process has not yet completed.

For more information on SASL authentication, see Chapter 13, “Using SASL Authentication.”

LDAP_SERVER_DOWN

This result code indicates that the LDAP API library cannot establish a connection (or lost the connection) with the LDAP server.

The LDAP API library sets this result code. If you have not established an initial connection with the server, verify that you have specified the correct hostname and port number and that the server is running.

If you have lost the connection to the server, see “Handling Failover” for instructions on reconnecting to the server.

LDAP_SIZELIMIT_EXCEEDED

This result code indicates that the maximum number of search results to return has been exceeded.

The size limit is specified in the search request. If you specify no size limit, the server will set the time limit.

When working with the Directory Server, keep in mind the following:

- If you are bound as the root DN and specify no size limit, the server enforces no size limit at all.
- If you are not bound as the root DN and specify no size limit, the server sets the size limit to the value specified by the `sizelimit` directive in the server’s `slapd.conf` configuration file.
- If the size limit that you specify exceeds the value specified by the `sizelimit` directive in the server’s `slapd.conf` configuration file, the server uses the size limit specified in the configuration file.

LDAP_SORT_CONTROL_MISSING

This result code indicates that server did not receive a required server-side sorting control.

The Directory Server 4.0 sends this result code back to the client if the server receives a search request with a "virtual list view" control but no server-side sorting control.

The "virtual list view" control requires a server-side sorting control. For more information on this control, see “Using the Virtual List View Control.”

Note that versions of the Directory Server prior to 4.0 do not support the "virtual list view" control.

LDAP_STRONG_AUTH_NOT_SUPPORTED

This result code is returned as the result of a bind operation. This code indicates that the server does not recognize or support the specified authentication method.

LDAP_STRONG_AUTH_REQUIRED

This result code indicates that a stronger method of authentication is required to perform the operation.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_SUCCESS

This result code indicates that the LDAP operation was successful.

LDAP_TIMELIMIT_EXCEEDED

This result code indicates that the time limit on a search operation has been exceeded.

The time limit is specified in the search request. If you specify no time limit, the server will set the time limit.

When working with the Directory Server, keep in mind the following:

- If you are bound as the root DN and specify no time limit, the server enforces no size limit at all.
- If you are not bound as the root DN and specify no time limit, the server sets the time limit to the value specified by the `timelimit` directive in the server's `slapd.conf` configuration file.
- If the time limit that you specify exceeds the value specified by the `timelimit` directive in the server's `slapd.conf` configuration file, the server uses the time limit specified in the configuration file.

LDAP_TIMEOUT

This result code indicates that the LDAP client timed out while waiting for a response from the server.

The LDAP API library sets this result code in the LDAP structure if the timeout period (for example, in a search request) has been exceeded and the server has not responded.

LDAP_TYPE_OR_VALUE_EXISTS

This result code indicates that the request attempted to add an attribute type or value that already exists.

The Directory Server sends this result code back to the client in the following situations:

- The request attempts to add values that already exist in the attribute.
- The request is adding an attribute to the schema of the server, but the OID of the attribute is already used by an object class in the schema.
- The request is adding an object class to the schema of the server, and one of the following occurs:
  - The object class already exists.
  - The OID of the object class is already used by another object class or an attribute in the schema.
  - The superior object class for this new object class does not exist.

LDAP_UNAVAILABLE

This result code indicates that the server is unavailable to perform the requested operation.

At this point in time, neither the LDAP API library nor the Directory Server return this result code.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION

This result code indicates that the specified control or matching rule is not supported by the server.

The Directory Server might send back this result code if the request includes an unsupported control or if the filter in the search request specifies an unsupported matching rule.

LDAP_UNDEFINED_TYPE

This result code indicates that the request specifies an undefined attribute type.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_UNWILLING_TO_PERFORM

This result code indicates that the server is unwilling to perform to requested operation.

The Directory Server sends this result code back to the client in the following situations:

- The client has logged in for the first time and needs to change its password, but the client is requesting to perform other LDAP operations.

  In this situation, the result code is accompanied by an expired password control. For details, see “Using Password Policy Controls.”

- The NT Synch Service is running, and an operation is "vetoed" by the service.
- The request is a modify DN request, and a "superior DN" is specified. (At this point in time, the Directory Server does not support the ability to use the modify DN operation to move an entry from one location in the directory tree to another location.)
- The database is in read-only mode, and the request attempts to write to the directory.
- The request is a delete request that attempts to delete the root DSE.
- The request is a modify DN request that attempts to modify the DN of the root DSE.
- The request is a modify request to modify the schema entry, and one of the following occurs:
  - The operation is `LDAP_MOD_REPLACE`. (The server does not allow you to replace schema entry attributes.)

- The request attempts to delete an object class that is the parent of another object class.
- The request attempts to delete a read-only object class or attribute.

- The server uses a database plug-in that does not implement the operation specified in the request. For example, if the database plug-in does not implement the add operation, sending an add request will return this result code.

LDAP_USER_CANCELLED

This result code indicates that the user cancelled the LDAP operation.

Currently, the Directory Server does not send this result code back to LDAP clients.

LDAP_USER_CANCELLED

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 that under the "ou=People,o=airius.com" subtree of the directory, "ou=People,o=airius.com" is the base DN.

For more information, see "Specifying the Base DN and Scope," on page 96.

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, see Chapter 14, "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 a directory 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.

For more information, see "Understanding DSEs," on page 207.

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, see Chapter 15, "Working with Extended Operations.

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.

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.

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.

root DSE An entry (a DSE) that is located at the root of the DIT.

For more information, see "Getting the Root DSE," on page 208.

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.)

search result reference See search reference.

server plug-in The Netscape Directory Server 3.0 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 3.0 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, see "Getting Schema Information," on page 213.

Index

E

F

G

H

I

L

M

N

O

P

R

S