



Operating System

Understanding LDAP

White Paper

Dan Thompson

Abstract

This paper describes the Light Weight Directory Access Protocol (LDAP), an open network protocol standard designed to provide access to distributed directories. LDAP provides a mechanism to query or modify information that exists in a directory information tree (DIT), which may contain a broad range of information about different types of objects such as users, printers, applications, and other network resources. This document presents information on the basic models used to describe LDAP, and describes the APIs used to expose the LDAP protocol.

© 2000 Microsoft Corporation. All rights reserved.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA

07/00

Contents

Introduction.....	1
LDAP Fundamentals	2
Origin	2
Informational Model	3
Schema	3
Entry	5
Attributes	5
Naming Model	5
Distinguished Name	5
Relative Distinguished Name	5
Functional Model	6
Authentication	6
Interrogation	6
Update	7
Security Model	8
Additional Concepts	10
Ports	10
Import and Export Methods	11
Synchronous Vs Asynchronous Operations	16
RootDSE	16
Administrative Limits and Query Policy	18
Referrals	21
Controls	25
Core LDAP APIs	29
Examples	29
The Ldp.exe Utility	29
Starting and Configuring Ldp.exe	30
Authentication	31
Ldap_open_s()	31
Ldap_bind_s()	32
Ldap_unbind_s()	35

Interrogation	36
Interrogation Examples	36
Ldap_search_s()	37
Ldap_compare_s()	77
Update	80
Ldap_add_s()	80
Ldap_modify_s()	85
Ldap_modrdn2_s()	94
Ldap_delete_s()	97
Interpreting LDAP Errors.....	100
LDAP Error Codes	100
Using a Network Trace to Interpret LDAP Errors	102
SUMMARY.....	103
Appendix A: Examples	104
Creating External Cross-References	104
Demo.vbs	106
Appendix B: RFC Reference	111
For More Information	112

Introduction

Light Weight Directory Access Protocol (LDAP) is an open network protocol standard designed to provide access to distributed directories. LDAP provides a mechanism for querying and modifying information that resides in a directory information tree (DIT). A directory information tree typically contains a broad range of information about different types of network objects including users, printers, applications, and other network resources. LDAP is described through four basic models: Information, Naming, Functional, and Security. The combination of these models introduces a nomenclature that describes entries and their attributes, and provides methods to query and manipulate their values.

This paper provides the following information:

- *LDAP Fundamentals* introduces the four models that describe LDAP and presents additional concepts that are relevant to the understanding of LDAP.
- *Core LDAP APIs* describes the fundamental APIs that are used to expose the LDAP protocol. Each API is reviewed from a programmatic point of view with respect to what is actually placed on the network wire. It is important to remember that LDAP is a network protocol standard, not a defined API standard. While there exist well-known APIs to access the LDAP protocol, each API is vendor-specific. The goal of this paper is not to analyze the different vendor implementations of LDAP APIs, but rather to study LDAP from a network protocol point of view.
- *Interpreting LDAP errors*. In addition to the RFC-defined errors that are returned by a Directory Server Agent (DSA) to a client, additional error information may be obtained from a network trace. How to interpret that error information is the focus of the final section.

Included with this document download is a file called Demo.vbs, a sample Visual Basic Scripting (.vbs) script that illustrates use of LDAP.

Also included are the following Network Monitor trace files associated with running the Demo.vbs sample:

- Authentication.cap: a network trace associated with establishing a connection using the Authentication APIs.
- Interrogation.cap: a network trace associated with directory search requests and responses using the Interrogation APIs.
- Update.cap: a network trace associated with updating the directory information tree using the Update APIs.

LDAP Fundamentals

This section discusses the origins of LDAP and the basic models used to describe LDAP. It also highlights additional LDAP topics that are necessary to be successful in an LDAP environment.

Origin

In today's distributed computing environment, directories are required to locate resources. Directories typically consist of databases and protocols. A directory database must have a schema that acts as the blue print for the various objects that exist in the directory, and it must be capable of being distributed across the network. Because a directory database is distributed across the network, specialized directory access protocols are required to query and manipulate the directory remotely. Directories and databases have many things in common such as the same central theme of allowing access to stored data. However, directories have special requirements that differentiate them from relational databases. Directories are designed to be 'read-mostly.' The distributed nature of directories makes them inherently better at read requests as opposed to frequent updates. Because the directory is distributed, there may exist entries or attributes that have values in different states due to the delayed replication propagation. Finally, directories lack the transactional semantics that are found in modern relational databases.

The need to query directories may be traced back to the days of X.500 and Directory Access Protocol (DAP). X.500 is an International Standards Organization (ISO) standard that defines directory entries in a hierarchical namespace of information. The namespace could be arranged into a Directory Information Tree that is composed of Directory Server Agents. A Directory Server Agent is a computer that actually hosts a network directory and functions as an LDAP server. The Directory Information Tree can be searched from client computers by using the Directory Access Protocol (DAP), which was built on top of the Open Systems Interconnection (OSI) protocol stack.

Light Weight Directory Access Protocol was designed to provide access to distributed directories. The objective of the LDAP designers was to provide a highly functional directory access and manipulation protocol that remained relatively simple. LDAP allows one to query and manipulate information that exists in the DIT. The DIT might include information about users (e-mail or phone numbers) or resources. The term *resources* broadly describes printers, Distributed File System shares, or SQL Server databases.

The key aspects of LDAP are the following:

- Protocol elements are carried directly over Internet Protocol (IP), as either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).
- Many protocol data elements are encoded as ordinary strings.
- A lightweight Basic Encoding Rules (BER) encoding method is used to encode all protocol elements.

The popularity of LDAP may be attributed to the fact that it has excelled in four different areas: features, standards, implementation, and APIs. The ability to authenticate, query, and manipulate directories are the features required of a directory access protocol. The LDAP protocol is an IETF (Internet Engineering Task Force) standard. The IETF has also developed standards for well-known protocols like Domain Name System (DNS), Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), and Hypertext Transfer Protocol (HTTP). Several parties including the University of Michigan, Netscape, and Microsoft have implemented the LDAP protocol. Finally, the APIs were defined via the information RFC 1823. It is important to note the distinction between the APIs and the protocol.

LDAP is described by a combination of the following models:

- An Informational model that describes the structure of information in a directory information tree.
- A Naming model that describes how information is organized and referenced.
- A Functional model that describes what can be done with the information.
- A Security model that describes how information is protected in the directory information tree.

Informational Model

The informational model was derived from the ISO X.500 standard for creating enterprise-level directories. The informational model as defined in RFC 1777 "Lightweight Directory Access Protocol" and RFC 2251 "Lightweight Directory Access Protocol (v3)" describes entries and attributes. Information about what a schema is has been included for the completeness of this discussion.

Schema

A schema acts as a blue print or a template for the directory. The schema provides a listing of classes and attributes from which all entries are derived. For an entry or attribute to exist in the directory information tree, it must adhere to the definitions that are described in the schema. The schema defines the available classes, attribute type definitions, and the syntax from which an entry can be derived.

Classes

A class is a category of objects that share a set of common characteristics. Each object in the directory is an instance of one or more classes in the schema. RFC 2252, section 4.4, states that object classes are defined in X.501. In general, every entry contains an *abstract* class ("top" or "alias"), at least one *structural* object class, and zero or more *auxiliary* object classes. Microsoft chose not to implement the abstract class "alias," which is used in some directory information trees as a pointer when objects have been moved. The alias class is not necessary within the

Microsoft directory information tree because Microsoft made objects rename-safe by attaching an attribute called a Globally Unique Identifier (GUID) to each object. A GUID is a 128-bit (16-byte) number generated by an algorithm designed to ensure its uniqueness. This algorithm is part of the Open Software Foundation (OSF) Distributed Computing Environment (DCE), which provides a set of standards for distributed computing.

Abstract classes serve as templates for structural classes. Abstract classes must not exist in the DSA as an entry.

Structural classes are derived from abstract classes; they may exist in the DSA as an entry. Structural classes inherit all the attributes that are associated with parent abstract classes.

Auxiliary classes allow an entry to inherit a specific set of attributes. Auxiliary classes work like include files. The '88' classes were defined before the 1993 X.500 standard and may be considered a legacy. If you look at the attribute **ObjectClass** for an entry, you can determine which classes were used to derive that entry. For example, the **ObjectClass** attribute of a user entry enumerates the following classes: **top**, **person**, **organizationalPerson**, and **user**. **Top**, **person**, and **organizationalPerson** are all abstract classes. **User** is the structural class from which the entry was actually created.

Attributes

Attributes are data items used to describe the classes defined in the schema. They are defined in the schema separately from the classes, which allows a single attribute definition to be applied to many classes. The attribute type is identified by a short descriptive name and an OID (object identifier). Attributes are defined in RFC 2252 "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions." Examples of attributes that are defined in RFC 2256 LDAPv3 Schema include:

- Common Name – CN (2.5.4.3)
- Organizational Unit – OU (2.5.4.11)
- ObjectClass (2.5.4.0)

Syntax

The syntax for an attribute defines the storage representation, byte ordering, and the matching rules for comparisons of property types. The syntax specifies whether the attribute value must be a string, a number, a unit of time, and so forth. Every attribute of every object is associated with exactly one syntax. RFC 2252 defines the syntax requirements. The syntaxes for use with LDAP are named by OIDs. Examples of syntaxes include:

- Distinguished Name (1.3.6.1.4.1.1466.115.121.1.12)
- UTC time (1.3.6.1.4.1.1466.115.121.1.53)

-
- Object Class Description (1.3.6.1.4.1.1466.115.121.1.37)

Entry

An entry is either a container or a leaf object of a specific structural class. Microsoft often refers to directory entries as 'objects'. An entry is composed of various attributes that are defined in the schema for the class from which it was derived. Entries must have a Relative Distinguished Name (RDN) that is unique among its siblings, and the concatenation of the RDNs or Distinguished Name (DN) must be unique in the directory information tree.

Attributes

Both container and leaf entries may have attributes. Attributes are defined in the schema and must obey the schema's definition. Microsoft often refers to attributes as *properties*. Attributes are composed of a name, an OID, and a value. The syntax for the attribute is defined in the schema. Attributes are classified as 'must' or 'may'. Attributes that classified as 'must' are mandatory attributes, and an entry must have the attribute to exist. 'May' attributes are optional for the entry. Finally, attributes may be either single or multiple value in nature.

Naming Model

The OSI directory model used distinguished name as the primary key for entries in the directory. The naming model is outlined briefly in RFCs 1777 and 2251. The LDAP naming model was further enumerated in RFC 1779 "A String Representation of Distinguished Names" and RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names."

Distinguished Name

Entries are arranged in the directory information tree based on their Distinguished Name. The Distinguished Name consists of a series of Relative Distinguished Names and serves as a primary key for an object in the directory information tree. Each naming component represents a branch in the directory information tree. A Distinguished Name is analogous to the absolute path name to a file in the Windows file system.

Following are examples of Distinguished Names. In these examples, cn means CommonName and dc means DomainComponent.

- **cn=Dan,dc=Microsoft,dc=com**
- **cn=tim,cn=mydomain,dc=com**

Relative Distinguished Name

Each component of the Distinguished Name is a Relative Distinguished Name. The Relative Distinguished Name is unique within its container, and is analogous to a

file name or directory in a file system. The RDN consists of an attribute type and a value, and is formatted as: **<rdncomp> ::= <attr> '=' <value>**.

Examples of RDNs are listed below, where OU stands for organizational unit:

- **cn=Dan**
- **ou= Sales**
- **dc= Microsoft**

Functional Model

The functional model consists of nine operations in three areas:

- *Authentication* allows the client to prove its identity to the DSA.
- *Interrogation* provides a method for the client to interrogate the directory information tree.
- *Update* defines a mechanism for the client to add or modify information in the directory information tree.

Authentication

The authentication model includes the following operations:

- **Open**. This command creates and initializes a connection block, then opens the connection to the DSA.
- **Bind**. This command initiates a protocol session to the DSA. After a session is established, a method of authentication is negotiated between the DSA and the client. When the client is authenticated by the DSA, the DSA returns a **Bind** response to the client.
- **Unbind**. This command terminates an LDAP session between the client and the DSA.

Interrogation

The interrogation model includes the following operations:

- **Search**. This operation is used to select entries from a specific region of the directory information tree based on customized criteria called a *search filter*. The following arguments are used to perform the search:
 - **Search base**. The distinguished name of the search base object. This defines the location in the directory from which to begin searching.
 - **Search scope**. Defines how deep to search within the search base. The following options are available:

Base or zero level. Searches the base object only.

One level. Searches objects immediately subordinate to the base object, but excludes the base object.

Subtree. Searches the entire subtree of which the base distinguished name is the topmost object, including that base object.

- **Filter.** Allows certain entries in the subtree and excludes others. The filter operators are listed in Table 1.
- **Selection.** Indicates what attributes to return from objects that match the filter.
- **Optional controls.** These affect how the search is processed.

LDAP search filters, as defined in RFC 2254, allow you to define specific search criteria, resulting in more efficient searches. For example, you might be interested in all the users whose surname is Smith, or you might want to find out all the team members who report to the manager named Mary Jones. The search filters are represented by UTF-8 strings.

- **Compare.** This operation returns a Boolean response based upon a comparison of an entry's attribute value.

Table 1 lists the LDAP Search filter operators.

Table 1. LDAP Filter Operators

LDAP Filter Operator	Description
=	Equal
~=	Approximately Equal
<=	Less than or equal to
>=	Greater than or equal to
&	AND
	OR
!	NOT

Update

The update model consists of the following operations:

- **Add.** This command creates an object in the directory information tree based on information provided by the client to the DSA. The information that is passed to the DSA must meet the conditions that are imposed on entry creation through the classes that are defined in the schema.
- **Modify.** This command allows the client to modify an entry's attributes.

Modification of attributes includes creating, modifying, and deleting the attributes.

- **Modify RDN.** This command provides a mechanism for an entry to be moved in the directory information tree. By modifying the RDN components of an entry, the entry is effectively moved to a new container within the DIT.
- **Delete.** This command provides a method for a client to remove an entry from the directory information tree. The success of an entry modification is dependent on the schema's constraints and the access permission of the client.

Security Model

The security model specifies how to access information in the directory in a secure manner. RFC 2251 "Lightweight Directory Access Protocol (v3)" states that Simple Authentication and Security Layer (SASL) mechanisms may be used with LDAP to provide association security services. RFC 2222 defines SASL.

The root DSE includes an attribute called **supportedSASLMechanisms** which is a list of the supported SASL security features.

RFC 2222 "Simple Authentication and Security Layer" describes a method for providing authentication support in connection-based protocols. To use this specification, protocols have to include a command to identify and authenticate users to a server, and to optionally negotiate protection of subsequent protocol interactions. If protection is negotiated, a security layer is inserted between the protocol and the connection. RFC 2222 explains how a protocol specifies such a command, and defines several mechanisms for use by the command as well as the protocol used for carrying a negotiated security layer over the connection.

The Windows 2000 Active Directory supports SASL mechanisms including Kerberos Version 5 and MS Negotiate. To see the supported SASL mechanisms, you can query the rootDSE. When you perform the query, the Active Directory should return the **supportedSASLMechanisms** attribute (and others). The rootDSE attribute value is: **supportedSASLMechanisms: GSSAPI; GSS-SPNEGO**, where GSSAPI = Kerberos, and GSS-SPNEGO = NT Negotiate (Kerberos, NT LAN Manager (NTLM), and so on).

The Generic Security Service Application Program Interface (GSS API), defined in RFC 2078, provides security services to callers in a generic manner that can be supported with a range of underlying mechanisms and technologies. This allows applications (at source level) to be ported to different environments. RFC 2078 defines GSS-API services and primitives at a level independent of underlying mechanism and programming language environment.

RFC 1510 "The Kerberos Network Authentication Service (V5)" defines an authentication process which provides a method for verifying the identities of

principals (workstation users and network servers) on an open network. For authentication purposes, clients use Kerberos tickets, which represent the client's network credentials. Clients obtain the tickets from the Kerberos Key Distribution Center (KDC), and they present these tickets when a network connection is established. Kerberos represents the client's identity by using the domain name, user name, and password.

The Windows 2000 security infrastructure also supports the following primary security protocols:

- Windows NT LAN Manager (NTLM) authentication protocol is provided to support Windows NT version 4.0 and earlier. NTLM will continue to be supported and used for pass-through network authentication, remote file access, and authenticated Remote Procedure Call (RPC) connections to previous versions of Windows NT.
- Distributed Password Authentication (DPA) is the shared secret authentication protocol that is used by many Internet membership organizations, such as MSN and CompuServe. This authentication protocol is part of Microsoft Commercial Internet System (MCIS) services and is specifically designed to allow users to use the same Internet membership password to connect to various Internet sites that are part of the same membership organization. The Internet content servers use the MCIS authentication service as a back end Internet service, and users can connect to multiple sites without reentering their passwords.

Table 2 lists the authentication methods that are supported by the Microsoft implementation of the LDAP API.

Table 2. Authentication Methods Supported by the LDAP API.

Authentication Method	Description	Credential
LDAP_AUTH_SIMPLE	Authentication with a simple clear-text password.	A string containing the user's password.
LDAP_AUTH_NTLM	Windows NT@ LAN Manager	An array of strings containing the domain name, the user name, and the encrypted password.
LDAP_AUTH_DPA	Distributed password authentication (used by Microsoft Membership System)	
LDAP_AUTH_NEGOTIATE	Generic security services (GSS) (Snego). Does not provide any authentication services, instead chooses the most appropriate authentication method from a list of available services and passes all authentication information on to that service. Use with Windows@ 2000	To log in as the current user, set the <i>dn</i> and <i>cred</i> parameters to NULL. To log in as another user, pass a pointer to a SEC_WINNT_AUTH_IDENTITY structure with the appropriate user name and password.
LDAP_AUTH_SSPI	This constant is obsolete and is included for backward compatibility only. Using this constant selects GSS (Snego) negotiation service.	

For more information on Windows 2000 security features, see the [Windows 2000 Security Services](#) section of the [Windows 2000 Server Web site](#).

Additional Concepts

This section introduces additional concepts that are relevant to your understanding of LDAP.

Ports

LDAP utilizes either a Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) to connect from the client to the DSA. This connection occurs over a socket. Table 2 below lists different end points that provide a range of functionality.

Table 2. LDAP Connection End Points

Function	Port
LDAP	389
LDAP Secure Sockets Layer (SSL)	636
Global Catalog (GC)	3268
Global Catalog Secure Sockets Layer	3269

Import and Export Methods

You can import and export objects in batch mode by using these Windows 2000 Server administrative tools: LDIFDE, CSVDE, and ADSI scripts. (These tools are installed automatically on all Windows 2000 servers.) You can use these tools to:

- Administer large numbers of objects like users, contacts, groups, servers, and printers in one operation.
- Export Active Directory data to other applications and services.
- Import information from other sources into Active Directory.

LDAP Data Interchange Format (LDIF)

LDAP Data Interchange Format (LDIF) is an Internet standard that defines a file format to perform batch import and export operations for directories that conform to LDAP standards. An LDIF file consists of a series of records that are divided by line separators. A record describes either a single directory entry or a set of modifications to a single directory entry and consists of one or more lines in the file.

The LDAP Data Interchange Format (LDIF) (file) format has a command-line utility called LDIF Directory Exchange (LDIFDE) that you can use to add, modify, rename, and delete directory objects. For example, you can use LDIFDE to extend the schema, export Active Directory user and group information to other applications or services, and to populate Active Directory with data from other directory services. You run the LDIFDE command on a computer running Windows 2000 Server, or you can copy the Ldifde.exe file to a computer running Windows 2000 Professional and then run the command.

The following is an example of an LDIF import file format:

```
dn: CN=picturePath,CN=Schema,CN=Configuration,DC=reskit,DC=com
changetype: add
attributeID: 1.2.840.113556.1.4.7000.125.19
attributeSyntax: 2.5.5.9
cn: picturePath
issinglevalued: TRUE
objectCategory: CN=Attribute-Schema,CN=Schema,CN=Configuration,
DC=reskit,DC=com
objectClass: attributeschema
oMSyntax: 2
```

You use LDIFDE, by typing **LDIFDE** at the command prompt, along with one of the parameters listed in Table 3.

Table 3. LDIFDE Parameters.

Parameter	Description
-i	Specifies import mode. If not specified, the default mode for LDIFDE is export.
-f filename	Specifies the import or export file name.
-s server name	Specifies the domain controller to perform the import or export operation.
-c string1 string2	Replace all occurrences of string1 with string2. This is generally used when importing data from one domain to another and the distinguished name of the export domain needs to be replaced with that of the import domain.
-v	Specifies verbose mode.
-j path	Sets the log file location. The default is the current path.
-t port number	Specifies an LDAP port number. The default LDAP port is 389. The global catalog port is 3268.
-d baseDN	Sets the distinguished name of the search base for data export.
-r LDAP-filter	Use to create a LDAP search filter for data export. For example, to export all users with your surname, the following filter could be used: -r "(&(objectClass=user)(sn=yoursurname))"
-p scope	Sets the search scope, which can be Base, OneLevel, or SubTree.
-l LDAP-attribute-list	Specifies the list of attributes to return in the results of an export query. If this parameter is omitted, all attributes are returned. For example, to retrieve only the distinguished name, common name, first name, surname, and telephone number of the returned objects, the following attribute list would be specified: -l "distinguishedName, cn, givenName, sn, telephone"

Parameter	Description
-o	List of attributes to be omitted from the results of an export query. This is typically used when exporting objects from Active Directory and then importing them into another LDAP-compliant directory. There may be some attributes that are not supported by the other directory, so these attributes can be omitted from the result set using this option. For example, to omit the objectGUID, whenChanged and whenCreated attributes, the following omission would be specified: -o "whenCreated, whenChanged, objectGUID"
-g	Do not perform paged searches.
-m	Omit attributes that only apply to Active Directory objects such as the ObjectGUID, objectSID, pwdLastSet and samAccountType attributes.
-n	Do not export binary values.
-k	Skip errors during the import operation and continue processing. Typical errors that may be skipped include those where the object already exists.
-a <i>user-distinguished-name password</i>	Sets the command to run using the supplied user distinguished name and password. (The default is to run using the credentials of the currently logged on user.) For example: a cn=yourname,dc=yourcompany,dc=com password
-b <i>username domain password</i>	Sets the command to run as <i>username domain password</i> . (The default is to run using the credentials of the currently logged on user.)For example: b yourusername yourcompanydomain password
-?	Displays online Help.

Exporting and Re-Importing Objects

Linked attributes contain information about links to a current object. During a normal export session, a parent object might be exported before its child object. On the re-import operation, if the parent object is added before the child object, the operation fails because the child object is not yet in the directory. However, when the **-m** parameter is used to export objects and re-import them into Active Directory, all entries that contain a linked attribute are appended to the end of the file. Moreover, the link addition is separated from the main object creation call so that the failure in membership addition does not cause the object creation to fail. The linked attribute is appended to the end of the file.

Active Directory has Security Accounts Manager (SAM) properties that are read-only because they are set by the system at the time the object is created. When the **-m** parameter is used to export objects and re-import them into Active Directory, all of the SAM attributes are ignored during the export operation. In that way, when the

entries are re-imported into Active Directory, they succeed because they do not contain any SAM information. An LDAP call to modify these entries fails.

Comma-Separated Value Directory Exchange (CSVDE)

You can perform bulk import and export of data to and from Active Directory by using files that store data in the Microsoft Comma-Separated Value (CSV) file format, also known as a .csv file. The CSV format consists of a simple text file with one or more lines of data where each value is separated by a comma. The text file contains entries where the initial entry is a comma-separated list of attribute names. Each subsequent entry represents a single object in the directory. Attribute values are delimited by commas or some other user-selected character.

When you view data in a .csv file, the values for multi-value attributes are expressed as a single value that is internally delimited by a second user-definable delimiting character (by default, \$). Attribute values are listed left to right in the order in which the attribute names are listed in the initial entry. Values are positional, and every entry must account for each attribute listing in the initial entry. The attribute names must be in the same order as the data in any line that follows the first line, as shown in the following example:

Cn,Firstname,Surname,Description

1stuserlogonname,1stuserfirstnameJohn,1stusersurname,Manager

2nduserlogonname,2nduserfirstname,2ndusersurname,President

The CSV file format is supported by many applications such as Microsoft® Excel that can read and save data in this format. Microsoft Exchange Server and other third-party administration tools can also import and export data by using the CSV format.

The CSV format has a command-line utility called CSV Directory Exchange (CSVDE) that you can use to add new objects only. You can run CSVDE on a Windows 2000–based server, or you can copy the Csvde.exe file to a Windows 2000–based workstation and then run the command.

You run CSVDE by typing **CSVDE** at the command prompt, along with any parameters you want to use. Table 4 lists the CSVDE parameters.

Table 4. CSVDE Parameters.

Parameter	Description
-i	Specifies import mode. If not specified, the default mode is export.
-f filename	Identifies the import or export file name.
-s server name	Specifies the domain controller to perform the import or export operation.
-c string1 string2	Replaces all occurrences of string1 with string2. This is generally used when importing data from one domain to another and the distinguished name of the export domain needs to be replaced with that of the import domain.

Parameter	Description
-v	Specifies verbose mode.
-j path	Sets the log file location. The default is the current path.
-t port number	Specifies an LDAP port number. The default LDAP port is 389, and the global catalog port is 3268.
-d baseDN	Sets the distinguished name of the search base for data export.
-r LDAP-filter	Creates an LDAP search filter for data export. For example, to export all users with your surname, you could use the following filter: -r "(&(objectClass=user)(sn=yoursurname))"
-p scope	Sets the search scope, which can be Base, OneLevel, or SubTree.
-l LDAP-attribute-list	Set the list of attributes to return in the results of an export query. If this parameter is omitted, all attributes are returned. For example, to retrieve only the distinguished name, common name, first name, surname and telephone number of the returned objects, the following attribute list would be specified: -l "distinguishedName, cn, givenName, sn, telephone"
-o	List of attributes to be omitted from the results of an export query. This is typically used when exporting objects from the Active Directory and then importing them into another LDAP-compliant directory. There may be some attributes that are not supported by the other directory, so these attributes may be omitted from the result set using this option. For example, to omit the objectGUID, whenChanged and whenCreated attributes the following omission would be specified: -o "whenCreated, whenChanged, objectGUID"
-g	Do not perform paged searches.
-m	Omits attributes that only apply to Active Directory objects such as the ObjectGUID, objectSID, pwdLastSet and samAccountType attributes.
-n	Do not export binary values.
-k	Skips errors during the import operation and continue processing. Typical errors that may be skipped include those where the object already exists.
-a user-distinguished-name password	Sets the command to run using the supplied user distinguished name and password. (The default is to run using the credentials of the currently logged on user.) For example: a cn=yourname,dc=yourcompany,dc=com password
-b username domain password	Sets the command to run as <i>username domain password</i> . (The default is to run using the credentials of the currently logged on user.) For example: b yourlogonname yourcompanydomain password
-?	Displays online Help.

Active Directory Service Interfaces (ADSI) Scripts

Active Directory Service Interfaces is a set of specifications for COM interfaces and objects. ADSI defines various types of objects that are used to represent various directory service items such as users, computers, printers, files, and so on. These

objects support one or more COM interfaces that provide access to information about the object itself (this is called metadata) and what the object represents. ADSI abstracts the capabilities of directory services from different network providers in a distributed computing environment and presents a single set of directory service interfaces for managing network resources. Administrators and developers can use ADSI services to enumerate and manage the resources in a directory service, regardless of which network environment contains the resource.

The Microsoft Active Directory Service Interfaces when combined with a scripting language such as VBScript or JScript, provide a system administrator with powerful tools for managing users, services, print queues, and other parts of a Windows NT® or Windows® 2000 network. Administrators can create ADSI-based scripts to automate tasks that would normally be accomplished using the Windows 2000 Administrative Tools or the Microsoft Management Console. ADSI scripting can also automate regular system maintenance tasks for hands-free operation, and it provides an easy interface for writing Web-based system management tools using Active Server Pages (ASP).

For more information about ADSI and scripting, refer to the [Microsoft Platform SDK](#).

Synchronous Vs Asynchronous Operations

Most LDAP operations can be performed either synchronously or asynchronously. A synchronous function call must return before the client application can continue running. Synchronous functions return an indication of the outcome of the operation, or an LDAP error code if the call fails. LDAP APIs that function synchronously have an `_s` appended to the function name, for example, `ldap_search_s`.

An asynchronous function allows the client to continue performing other tasks, including making new requests to the server or processing the results of a search. Both synchronous and asynchronous functions return a message ID for the initiated operation. However, an asynchronous function uses the message ID to request the results.

RootDSE

The RootDSE is a standard attribute defined in the LDAP version 3.0 specification. The RootDSE contains information about the directory server, including its capabilities and configuration. The search response contains a standard set of information that is defined in RFC 2251 "Lightweight Directory Access Protocol (v3)." The LDAP protocol assumes there are one or more servers that jointly provide access to a Directory Information Tree. At the root of the Directory Information Tree is a DSA-Specific Entry (DSE), which is not part of any naming context. Each server has different attribute values in the root DSE.

The root DSE publishes information about the LDAP server including which LDAP versions it supports, any supported Simple Authentication and Security Layer (SASL) mechanisms, supported controls as well as the DN for its

subschemaSubentry attribute. In addition to server information, operational attributes may be exposed that allow for extended administration functionality. You can retrieve rootDSE data from an LDAP v3 server by doing a base-level search with a null **BaseDN** and with filter **ObjectClass=***.

Section 5.2 of RFC 2252 defines a set of rootDSE attributes that should be published by LDAP v3 servers that support them. In addition, Section 3.4 of RFC 2251 adds the **subschemaSubentry** attribute, making a total of seven standard attributes published in the rootDSE section of an LDAP v3 server. These core attributes are defined as follows:

- **namingContexts**: The values of this attribute correspond to naming contexts which this server masters or shadows. If the server believes it contains the entire directory, the attribute has a single value, an empty string (indicating the null DN of the root). This attribute allows a client to choose suitable base objects for searching when it has contacted a server.
- **subschemaSubentry**: The value of this attribute is the name of a subschema entry (or subentry if the server is based on X.500(93)) in which the server makes available attributes specifying the schema. Supported attributes are exposed in the **attributeTypes** property and supported classes in the **objectClasses** property. The **subschemaSubentry** property and the subschema are defined in LDAP v3.
- **altServer**: The values of this attribute are Uniform Resource Locators (URLs) of other servers that may be contacted when this server becomes unavailable. If the server does not know of other servers that could be used for this purpose, this attribute is absent. Clients may cache this information in case their preferred LDAP server later becomes unavailable.
- **supportedExtension**: The values of this attribute are Object Identifiers (OIDs) which identify the supported extended operations that the server supports. If the server does not support any extensions, this attribute is absent.
- **supportedControl**: The values of this attribute are the Object Identifiers (OIDs) which identify the controls that the server supports. If the server does not support any controls, this attribute is absent.
- **supportedSASLMechanisms**: The values of this attribute are the names of supported SASL mechanisms that the server supports. If the server does not support any mechanisms, this attribute is absent. By default, Generic Security Service Application Program Interface (GSSAPI) is supported.
- **supportedLDAPVersion**: The values of this attribute are the versions of the LDAP protocol that the server implements.

Active Directory also supports the following informational attributes:

- **currentTime:** The current time based on Zulu time in the format xxxx(year)xx(month)xx(day)xxxxxx.x(hours,minutes,seconds military time)'Z'
- **dsServiceName:** NTDS Settings.
- **defaultNamingContext:** This is the default NC for a particular server. By default, this is the DN for the domain of which this directory server is a member.
- **schemaNamingContext:** DN for the Enterprise schema Naming Context.
- **configurationNamingContext:** DN Enterprise Configuration Naming Context.
- **rootDomainNamingContext:** This is the DN for the root of the Domain for which this server is a DC.
- **supportedLDAPPolicies:** Supported LDAP management policies.
- **highestCommittedUSN:** Highest USN committed to the database on this server.
- **dnsHostName:** The DNS name of this DC.
- **ldapServiceName:** Service Principal Name (SPN) for the LDAP server. This is used for mutual authentication.
- **serverName:** DN for the server object for this directory server as defined in the Configuration container.
- **supportedCapabilities:** The values of this attribute are Object Identifiers (OIDs) identifying the supported capabilities of the server. Object Identifiers are unique numeric values that are issued by various "Issuing Authorities" to identify data elements, syntaxes, and various other parts of distributed applications.

Administrative Limits and Query Policy

Support for LDAP v3 extensions for querying, paging, and sorting places demands on the memory and computational resources of the Active Directory server. It is prudent practice to perform load balance testing on LDAP servers before you deploy them. Only then can you develop a set of baseline measurements from which to make adjustments.

You can place limits on the server resources that are available to clients requesting LDAP queries, paged result sets, and sorted result sets. These limits constitute the LDAP query policy; they are stored as a multi-value attribute on query policy objects. Because workload and resources of a given server vary, the query policy is

configurable at the server level.

Query policy applies to the following LDAP query-related operations:

- **Search.** This is the basic query operation. An LDAP search might cover a small part of a single directory service store or span every directory service store in the forest (and beyond, through support for virtual containers). A search can generate a significant amount of disk activity, take a long time, and return a large volume of data.
- **Search with Paged Results.** Because a search can return a large volume of data, the client can ask the server to hold the result set and return it in "pages." (See "[Paging Search Results](#)" later in this document.) The server must hold the result set until the client releases it or unbinds.
- **Search with Sorted Results.** A client can request a result set in a particular order. Sorting requires storage and CPU cycles at the server. (See "[Sorting Search Results](#)" later in this document.) The resources consumed are directly proportional to the size of the result set.
- **Search with Replication.** The administrator can specify the maximum number of attribute values that can be returned per request.
- **Change Notify.** A client can request change notification on particular objects in the directory. The mechanism used to post a change notify request is the asynchronous LDAP query.

Query policy objects are stored in the container **cn=Query-Policies,cn=Directory Service,cn=Windows NT,cn=Services** in the configuration partition. In the absence of any other assigned policies, all domain controllers use the default query policy. If a site policy is assigned, the domain controller uses this policy. If a specific policy has been assigned to a domain controller, this policy takes precedence over any site policy.

Tools

Two tools are available for working with LDAP query policy information:

- The Windows 2000 Ntdsutil tool. You can use this tool to view or modify the query policy of a domain controller. Ntdsutil can be used to repair, check, compact, move, and dump the directory database files. It also lists site, domains and server information, manages operations masters, performs authoritative restore, create domains.

- The MODIFYLDAP.VBS script. You can use this script to create, delete, assign, or modify query policy objects. This script can be installed from the Support\Reskit directory on the *Windows® 2000 Resource Kit* companion CD.

For example, you can type the following at the command prompt:

```
CSCRIPT MODIFYLDAP.VBS /C /O:"New Query Policy" MaxConn:20 /U:NTDEVAdministrator /W:password
```

The administrative limits and values can be viewed by using the Ntdsutil command-line tool. Table 5 shows the administrative limits that are in effect for the default query policy.

Table 5. Default query policy settings table.

LDAP Administrative Limits	Default Value	Description and search behavior
InitRecvTimeout	120	Initial Receive Timeout. The maximum time in seconds that the server waits for the initial request before the connection closes. If a connection is idle for more than the stated limit, the LDAP server returns a LDAP disconnect notification and closes the connection.
MaxConnections	5000	Maximum Connections. The maximum number of concurrent LDAP connections allowed on the server. If the stated limit is reached, the LDAP server returns a LDAP disconnect notification and closes the connection.
MaxConnIdleTime	900	Maximum Connection Idle Time. The maximum time in seconds that the client is allowed to be idle before the connection is closed. If a connection is idle for more than the stated limit, the LDAP server returns a LDAP disconnect notification and closes the connection.
MaxActiveQueries	20	Maximum Active Queries. The maximum number of concurrent search operations allowed on the server. When the stated limit is reached, the LDAP server returns a busy notification.
MaxNotificationPerConn	5	Maximum Notifications per Connection. The maximum number of concurrent notification requests allowed per connection on the server. When the stated limit is reached, the server returns a busy notification.
MaxPageSize	1000	Maximum Page Size. The largest page size allowed by the server. The server returns the number of rows specified by MaxPageSize. If the paged results were requested, the client can retrieve additional pages until all results are returned.

LDAP Administrative Limits	Default Value	Description and search behavior
MaxQueryDuration	120	Maximum Query Duration. The maximum elapsed time (in seconds) allowed for a query to complete. If paged results are requested, the client can continue the query if the timer expires before the query completes. When the stated limit is reached, the server returns the <code>timeLimitExceeded</code> error.
MaxReceiveBuffer	10485760	Maximum Receive Buffer. The maximum size LDAP request in bytes that the server will attempt to process. If the server receives a request that is larger than this value, it will close the connection.
MaxTempTableSize	10000	Maximum Temporary Table Size. The upper limit, in candidate objects, on the temporary table. If the temporary table maximum limit is reached by an "OR" query optimization, the optimization is abandoned and replaced with a direct table scan.
MaxResultSetSize	262144	Maximum Result Set Storage. The maximum storage that the server can hold for all paged result sets. If the stated limit is reached, the oldest result sets are discarded.
MaxPoolThreads	4	Per Processor Asynchronous Thread Queue (ATQ) Threads. The number of threads allocated by ATQ per processor. This value is sent as an advisory notification to ATQ. ATQ decides whether to use it or not. Note: If it takes a long time to bind, increase the count to 6 or 8.
MaxDatagramRecvSize	1024	Maximum Receive Datagram Size. The maximum size of datagrams that can be received by the server. The server pre-allocates datagram buffers and cannot receive datagrams with a size larger than the stated limit.

Referrals

A directory system agent (DSA) manages the directory hierarchy information (referred to as *knowledge*) which it receives from the database layer. The DSA is responsible for cross-references of Active Directory domain objects within the hierarchy, and also out to other domain hierarchies. When a requested object exists in the directory but is not present on the contacted domain controller, name resolution depends on the given domain controller's knowledge of directory partitioning. In a partitioned directory, by definition, the entire directory is not

necessarily available on any one domain controller.

An LDAP referral is used by a domain controller to inform a client application that it does not have a copy of a requested object (that is, it does not hold the section of the directory tree where that object would be, if it exists). It also gives the client a location that is more likely to hold the object, which the client then uses as the basis for a DNS search for a domain controller. Ideally, referrals always reference a domain controller that indeed holds the object. However, it is possible for the referred-to domain controller to generate yet another referral, although it should not take long to discover that the object does not exist and to inform the client. Active Directory returns referrals in accordance with RFC 2251.

Every domain controller has information about the other domains in the forest in the domain controller's Configuration container. When an operation in Active Directory requires action on objects that might exist in the forest but are not located in the particular domain that is stored on a domain controller, that domain controller must generate a message to the client describing where to go to continue this action; that is, the client is "referred" to a domain controller that is presumed to hold the requested object. Without needing to know the name or location of the child domain, clients can query the root domain and reach the appropriate domain controller by being referred there. Two cases generate this type of domain controller response:

- The base distinguished name of the operation is not in this directory, but the domain controller has knowledge of another LDAP directory where it might be found (an "external referral").
- The base distinguished name of the operation is in this directory, but the operation requires proceeding into portions of the directory tree that are not stored on this domain controller (a "subordinate referral").

Every domain controller contains information (called knowledge) about how the directory is partitioned; this information can be used in combination with DNS to find the correct Active Directory domain. Referrals to other domain controllers can be generated by the DSA according to the information that Active Directory stores about the directory partitions. Active Directory stores information about the existence and location of directory partitions, including the names of the directory partitions, which server is holding read-only copies (partial directory partitions stored on Global Catalog servers), and which server is holding writable copies (full directory partitions).

Three kinds of knowledge references exist:

- A *subordinate reference* is knowledge of a directory partition or partitions directly below a directory partition held by this domain controller.
- A *cross reference* is knowledge of one directory partition, stored in a cross-reference object. On a given domain controller, the combination of all cross references provides knowledge of all directory partitions in the forest, regardless of their location in the directory tree. The state of cross-

reference knowledge at any given time is subject to the effects of replication latency.

- A *superior reference* is knowledge about a referral location that is used when the domain controller has no knowledge of the search base.

Knowledge references form the glue that holds the pieces of the distributed directory together. Because Active Directory is partitioned in directory partitions, either all objects in a directory partition are present on a given domain controller or no objects in the directory partition are present on the domain controller. For this reason, references have the effect of linking the partitions together, which allows operations such as searches to span multiple partitions.

In Active Directory, referrals are generated when the directory is asked to locate an object where, based on the position at which the search begins, no copy exists in a local directory partition. When Active Directory can definitively determine that no such object exists in the directory (rather than that it might exist but no copy exists here), instead of sending a referral, the directory returns an error that indicates to the client that no such object exists in the forest.

Subordinate References. When a search is requested, the domain controller searches all objects at or below the search base, within the directory partition the domain controller holds. If a subtree search has a search base that includes child partitions, the domain controller uses subordinate references to return referrals (called subordinate referrals) to these partitions. Subordinate referrals are returned as part of the data returned from the base distinguished name partition, and they contain the distinguished name of the subordinate directory partition and the access point to which queries can be referred. An access point consists of a DNS name and a port number, which is the information that is required to contact a specific LDAP server. Access points are generated from information on the cross-reference object.

Cross-references. These references are stored as directory objects of the class **crossRef** that identify the existence and location of all directory partitions, regardless of location in the directory tree. Cross references enable every domain controller to be aware of all directory partitions in the forest, not just the partitions that it holds. Because these objects are stored in the **Configuration** container, the knowledge they store is replicated to every domain controller in the forest. Values for the following attributes are required for each cross-reference:

- **nCName** is the distinguished name of the directory partition that the **crossRef** object references. (NC stands for *naming context*, which is a synonym for *directory partition*.) The combination of all of the **nCName** properties in the forest defines the entire directory tree, including the subordinate and superior relationships between partitions.

-
- **dnsRoot** identifies the DNS name of the domain where servers that store the given directory partition can be reached. This value can also be a DNS host name.

Cross-reference objects are used to generate referrals to other directory partitions and to foreign directories. Cross-reference objects are created in two ways:

- Internally by the system to refer to known locations within the forest.
- Externally by administrators to refer to locations that are external to the forest.

An internal cross-reference is an object that is created by the system. For every directory partition in a forest, there is an internal cross reference object in the **Partitions** container (cn=Partitions,cn=Configuration,dc=ForestRootDomain). When you create a new forest, the Active Directory Installation wizard creates three directory partitions: the first domain directory partition, the configuration directory partition, and the schema directory partition. For each of these partitions, a cross reference object is created automatically. Thereafter, when a new domain is created in the forest, another directory partition is created and the respective cross reference object is created. Because these cross reference objects are located in the **Configuration** container, they are replicated to every domain controller in the forest, and thus every domain controller has knowledge of the name of every partition in the forest (as well as their superior and subordinate relationships to each other). By virtue of this knowledge, any domain controller can generate referrals to any other domain in the forest, as well as to the schema and configuration directory partitions.

An external cross reference is a cross-reference object that can be created manually to provide the location of an object that is not stored in the forest. If your LDAP clients will submit operations for an external portion of the global LDAP namespace against servers in your forest, and you want your forest's servers to refer the client to the correct location, you can create a cross reference object for that directory in the Partitions container. External cross references are used in two ways:

- To reference foreign directories by their disjoint directory name (a name that is not contiguous with the name of this directory tree). In this case, when you create the cross reference, you create a foreign container that is not a child of any object in the directory.
- To reference foreign directories by a name that is within the Active Directory namespace (a name that is contiguous with the name of this directory tree). In this case, when you create the cross reference, you create a virtual container that is a child of a real directory object. In both cases, a non-instantiated, invisible "container" is created in the directory.

Superior References. A superior reference is the distinguished name of a directory partition that is stored in the **superiorDNSRoot** attribute on the **crossRef** object for the forest root domain (the first domain created in the forest). A domain controller uses its superior reference to construct a referral only when a search base does not match any directory partition defined by the cross-reference objects. A superior reference contains no directory tree information; it consists of only an access point to which otherwise unanswerable queries can be referred. By default, **superiorDNSRoot** does not store a value, but the directory uses the **DC=** components of the search base distinguished name to construct the equivalent of a superior referral. You can use the value in **superiorDNSRoot** attribute to define a location to send all queries that cannot be resolved.

Controls

RFC 1823, which defines the LDAP API for LDAP v3, is being updated to include support for new LDAP v3 controls. Windows 2000 supports several new controls that extend the functionality of the LDAP protocol. The LDAP API supports server controls and client controls. Server controls can be sent to a server or returned to the client with any LDAP message. Client controls affect the behavior of the LDAP API only and are never sent to a server.

You can use the Ldp.exe utility to implement these controls provided you know the control object identifier (OID). You can view all supported controls and their OIDs by reading the **supportedControls** property on the RootDSE for the domain controller LDP.exe. The OIDs for each control are designated in LDAP as the control type (**controlType**).

Tree Delete (controlType 1.2.840.113556.1.4.805)

When **Tree Delete** is used with a delete request (**DelRequest**) message, it allows a client to delete an entire subtree of a container object. When this control is invoked, the server checks to see that the user has permission to invoke the **Tree Delete** operation. In addition, other restraints on containers can override an attempt to delete it. For example, a deletion cannot cross a directory partition (or naming context) boundary.

Directory Synchronization (controlType 1.2.840.113556.1.4.841)

Directory Synchronization enables the Active Directory Read Provider to extract the latest changes from the Active Directory replication module. The Active Directory Read Provider is an LDAP-based provider that is responsible for reading Active Directory changes. Because the partition is the unit of replication, the Directory synchronization control must be invoked on the root of a partition.

Cross-Domain Move (controlType 1.2.840.113556.1.4.521)

Cross-Domain Move allows an object to be moved from one domain to another. The operation is used in conjunction with the LDAP ModifyDN request. In effect, the object is renamed to exist in a different domain.

Show Deleted Object (controlType 1.2.840.113556.1.4.417)

When an Active Directory object is deleted, it is stored for a configurable period of time to allow replication of the deletion to occur. The **Show Deleted Object** control allows objects that have been deleted but not yet placed in garbage collection to be viewed in Active Directory, along with objects that are not marked for deletion. These deleted objects, which are normally not visible after they are deleted (that is, the **isDeleted** attribute has a value of **TRUE**), can be viewed when you use **Show Deleted Object** in conjunction with search commands. If you want only the objects that are marked for deletion (also called "tombstones") to appear, add **isDeleted=true** to your filter.

Attribute Range Option (controlType 1.2.840.113556.1.4.802)

The LDAP protocol reads a multivalue attribute as a single entity, which can be inconvenient in the time that it takes when the number of values is large or, in some cases, makes reading the attribute impossible. The **Attribute Range Option** can be specified as part of an attribute description to retrieve the values of a multivalue attribute incrementally. An attribute description includes an attribute type (for example, **member**) and a list of options, one of which can be the **Range** option. When the **Range** option is presented in a **searchRequest** message, it specifies a zero-relative range of elements (for example, 0-9) to be retrieved. By specifying the **Range=** option followed by a range specifier, only the number of values in that range are retrieved.

Return Extended Distinguished Names (controlType 1.2.840.113556.1.4.529)

If an application needs to retain a long-term reference to a directory object, such as a user, a computer, or an organizational unit, it can use this control to ensure that the reference does not become obsolete. For example, distinguished names can change after a rename, move, or delete. However, the object's GUID never changes. This control returns a special kind of distinguished name that includes the GUID. If an application stores this extended distinguished name instead of the regular distinguished name, then the reference is always current.

Windows 2000 Supported Capabilities (controlType 1.2.840.113556.1.4.800)

This is published in the RootDSE under the **supportedCapabilities** attributes. It indicates that this is a Windows 2000 server and has all the capabilities associated with a Windows 2000 Server, including Ambiguous Name Resolution (ANR). It can be thought of as a version number for the directory service.

Verify Distinguished Name Server (controlType 1.2.840.113556.1.4.1338)

This control is used by the domain controller to determine the server necessary to verify that a distinguished name exists. For example, when you want to add a reference to an object from one domain that is actually stored in another domain, you can use this control.

Do Not Generate Referrals (controlType 1.2.840.113556.1.4.1339)

This control tells the domain controller not to generate LDAP referrals when trying to locate an object. This can be used to improve the performance of a search. For example, if you are not going to be generating referrals, and you only want to search for objects in a particular domain, using this control will save server overhead costs.

Server Search Operations (controlType 1.2.840.113556.1.4.1340)

This control allows the client to pass in flags to control various search behaviors. It can be used as an all purpose search control. Flags are passed in with this control to turn search options on or off. For example, the **Do Not Generate Referrals** control is the same as the **Server Search Operations** control with the **SERVER_SEARCH_FLAG_DOMAIN_SCOPE=0x1** flag, which is one of only two flags that are currently implemented.

The other flag, **SERVER_SEARCH_FLAG_PHANTOM_ROOT=0x2**, allows searches to specify a base distinguished name that is not actually in the forest of interest and still obtain valid results. For example, if the distinguished name of a particular root domain is DC=sales,DC=mydomain,DC=com a client can specify a base distinguished name of DC=com for a search and the directory returns everything it is aware of below DC=com. In this case, it could only return objects in the DC=reskit,DC=com domain directory partition. Without this flag the server would return an error.

Paging Search Results (controlType 1.2.840.113556.1.4.319)

Paging allows the client to retrieve a result set in small pieces. For example, when an LDAP client is connected to the server across a slow link or if the result set is expected to be very large, results can be requested in pages of a specific size. The new control extension for simple paging has options that allow the client to control the rate at which an LDAP server returns the results of an LDAP search operation.

An LDAP client application can specify a paged results control (**pagedResultsControl**) with a search request (**searchRequest**) that specifies a desired page size, which means the number of entries to be returned in a single response. Each time the server returns a set of results to the client, the server includes the **pagedResultsControl** control in the **searchResultDone** message. In the message returned to the client, if the server has the ability, it returns an estimate of the total number of entries in the entire result set in the form of a cookie. If the value in the cookie indicates more pages, the client then sends another search request, including the cookie and a new message ID, and the process continues until the **searchResultDone** message returns a value of 0 in the cookie.

Sorting Search Results (controlType 1.2.840.113556.1.4.473)

Sorting can be requested when the LDAP client needs search results to be sorted but is not able to perform the sort itself. The **Request** control (**sortKeyRequestControl**) allows the client to give the server the information

required to sort the result set. The **Request** control is included in the **searchRequest** message to the server. The **Response** (**sortKeyResponseControl**) control adds the functionality that enables the server to respond to the sort request. The **Response** control is included in the **searchResultDone** message from the server.

The **sortKeyRequestControl** specifies one or more attribute types and matching rules for the results returned by a search request, as well as a flag indicating forward or reverse order. The server should return all results for the search request in the order specified by the sort keys.

Core LDAP APIs

This section describes the authentication, interrogation, and update APIs.

Examples

The examples included in this section are based on a hypothetical organizational unit (OU) named Demo. You create the Demo OU by running the Demo.vbs script that was included when you downloaded this document and its associated files (the .cap files). Note that the domain component in the examples is **dc=mydomain,dc=com**. The domain component will adapt to your environment and will reflect the naming conventions that you have implemented.

Along with the Demo.vbs script sample and this document, the file download included three network trace files: authentication.cap, interrogation.cap, and update.cap. These files represent network trace captures that are associated with running the Ldp.exe utility, described below, to perform the various Active Directory operations described in each API section in this document. To view the .cap files, you run Network Monitor (Netmon.exe), click **Open** on the **File** menu, and then select the .cap file you want to open in the **Open** dialog box. (In Windows 2000 Server, Netmon.exe is installed in the Windows directory in the \System32\Netmon folder, for example, c:\Winnt\System32\Netmon.)

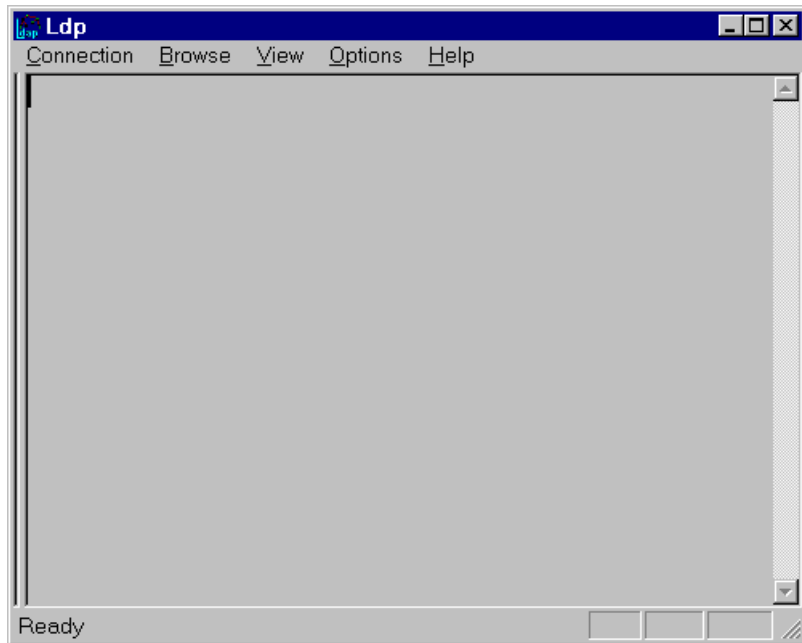
For example, to illustrate what network traffic occurs when a call to the **ldap_open_s()** authentication function is made to initiate a connection with the server, the Ldp.exe utility, described below, is started and connection options are entered (name of the server and port number). The process of starting and configuring Ldp.exe is described in a procedure which is then followed by a network trace capture that shows actual use of the **ldap_open_s()** function call that results from configuring this connection in Ldp.exe. An example of the output from Ldp.exe associated with this connection process is shown below.

```
ld = ldap_open("mydomain.com", 389);  
Established connection to mydomain.com.
```

The Ldp.exe Utility

As mentioned previously, the examples included in this section use the Ldp.exe utility to configure various LDAP operations to be performed against Active Directory. Ldp.exe is a Windows 2000 Resource Kit utility that is also included in the Windows 2000 Server compact disc in the Support\Reskit folder. Ldp.exe is a graphical tool that you can use to perform LDAP operations, such as connect, bind, search, modify, add, and delete, against any LDAP-compatible directory, such as the Active Directory. For troubleshooting purposes, administrators can use Ldp.exe to view objects stored in the Active Directory along with their metadata, such as security descriptors and replication metadata.

The Ldp window is shown below.



The Ldp window.

For more information about Ldp.exe and other Active Directory support tools, refer to the Windows 2000 Resource Kit. A Web version of the [Windows 2000 Server Resource Kit](#) will be available by subscription from the [Microsoft Windows 2000 Web Site](#).

Starting and Configuring Ldp.exe

This section presents instructions for starting and configuring Ldp.exe.

To start Ldp on the client

- Click **Start**, click **Run**, and type **Ldp**. Or double-click Ldp.exe.

The Auto Distinguished Name base query should be configured to **False**.

To configure the base DN option

1. On the **Options** menu, click **General**.
2. In the **General Options** dialog box, clear the **Auto base DN query** check box.

Authentication

The authentication APIs are used when a client connects to a DSA. Connection to the DSA occurs whenever the client requests information from the DIT. The client may be searching for a resource, modifying information in the DIT, or running a directory aware application. Note that, depending on your environment, different authentication mechanisms may be negotiated. This section describes the **Ldap_open_s()**, **Ldap_bind_s()**, and **Ldap_unbind_s()** functions.

Ldap_open_s()

The **Ldap_open** function creates and initializes a connection block, and then opens the connection to an LDAP server. If this function succeeds, it returns a session handle as a pointer to an LDAP data structure; if it fails, it returns a null pointer. You can retrieve an error code by using the **LdapGetLastError** function.

Syntax

```
WINLDAPAPI LDAP * LDAPAPI ldap_open(  
    PCHAR HostName  
    ULONG PortNumber  
);
```

Parameters

HostName

A list of host names or dotted strings representing the IP address of LDAP server hosts. Each host name in the list can include an optional port number, which is separated from the host itself with a colon (:) character. The hosts are tried in the order listed, stopping with the first successful connection. Note that only **ldap_open** attempts to make the connection before returning to the caller. The function **ldap_init** does not connect to the LDAP server.

PortNumber

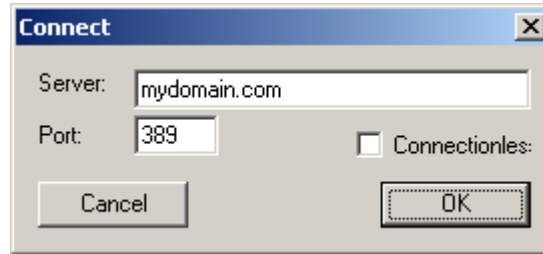
Contains the TCP port number to which to connect. The default LDAP port, 389, can be obtained by supplying the constant **LDAP_PORT**. If a host includes a port number then this parameter is ignored.

Example: Ldap_open

Use of the **Ldap_open** function is illustrated in the following example using **Ldp.exe**. A network trace of this process is included after the instructions for setting connection options.

To specify connection options

1. In the Ldp window, on the **Connection** menu, click **Connect**.
2. In the **Connect** dialog box, in the **Server** text box, type the domain name, and click **OK**.



The Ldp window shows the following information:

```
ld = ldap_open("mydomain.com", 389);  
Established connection to mydomain.com.
```

From the network trace capture, one can see the connection block established. This information is shown in Frame 1, included below. See the Authentication.cap file (included as part of the files you downloaded with this document) for details.

Frame 1

```
TCP: ...S., len: 0, seq:1171961004-1171961004, ack: 0,  
win:16384, src: 1148 dst: 389 // Note: Connected to LDAP port 389
```

Frame 2

```
TCP: .A..S., len: 0, seq:2701668798-2701668798, ack:1171961005,  
win:17520, src: 389 dst: 1148
```

Frame 3

```
TCP: .A...., len: 0, seq:1171961005-1171961005, ack:2701668799,  
win:17520, src: 1148 dst: 389
```

For more information about a TCP connection block, refer to the Microsoft Knowledge Base article [Q169292 The Basics of Reading TCP/IP Traces](http://support.microsoft.com/kb/q169292).

Ldap_bind_s()

The **Ldap_bind_s()** function is used to authenticate a client to the LDAP server. If the function succeeds, it returns the message ID of the operation initiated.

To identify a client to the directory server, the bind operation provides a distinguished name and some type of authentication credential, such as a

password. The exact credentials depend on the authentication method being used. If NULL is passed in for the credentials with **Ldap_bind_s()** (**non-simple**), the credentials of the current user or service are used. If a simple bind method is specified (as in **Ldap_simple_bind_s**), it is equivalent to a NULL plain text password.

Syntax

```
#include <winldap.h> WINLDAPAPI ULONG LDAPAPI ldap_bind_s(  
    [in]LDAP *ld,  
    [in]PCHAR dn,  
    [in]PCHAR cred,  
    [in]ULONG method  
);
```

Parameters

ld

The session handle.

dn

The distinguished name of the entry to bind as.

cred

The credentials to use for authentication. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the mechanism parameter.

method

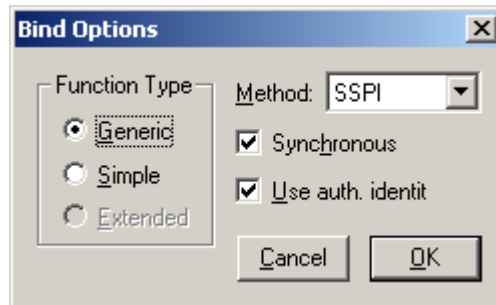
Indicates the authentication method to use.

Example: Ldap_bind_s()

The following example uses ldp.exe and a network trace to illustrate use of the **Ldap_bind_s()** function.

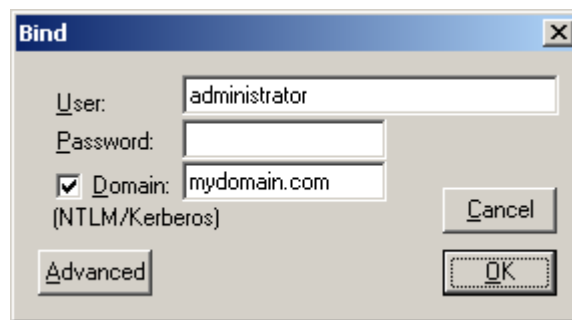
To configure binding options for the LDAP session

1. In the Ldp window, on the **Options** menu, click **Bind**.
2. In the **Bind Options** dialog box specify the settings to use, as shown below:



To bind to the DSA with the LDAP client

1. In the Ldp window, on the **Connection** menu, click **Bind**.
2. In the **Bind** dialog box, type **administrator** the **User** field and a domain name in the **Domain** field (mydomain.com, for example).



The ldp window shows this information:

```
res = ldap_bind_s(ld, NULL, &NtAuthIdentity, 1158); // v.3
{NtAuthIdentity: User='administrator'; Pwd= <unavailable>; domain =
'mydomain.com'.}
Authenticated as dn:'administrator'.
```

From a network trace, one can see the bind request (**BindRequest**) and bind responses (**BindResponse**) from frames 11 and 14. Note the Message ID that is used to track the request and response.

Frame 11

```
LDAP: ProtocolOp: BindRequest (0)
  LDAP: MessageID // Note the Message ID in the Data = 0x0E
  LDAP: ProtocolOp = BindRequest // Value in Data = 0x60
  LDAP: Version = 3 (0x3)
  LDAP: Authentication Type = Sasl
    LDAP: Sasl Mechanism = GSS-SPNEGO
    LDAP: Sasl Credentials
```

Frame 14

```
LDAP: ProtocolOp: BindResponse (1)
  LDAP: MessageID // Note the Message ID in the Data = 0x0E
  LDAP: ProtocolOp = BindResponse // Value in Data = 0x61
  LDAP: Result Code = Success
  LDAP: Sasl Mechanism = á
```

Ldap_unbind_s()

The **ldap_unbind_s** function frees all resources associated with an LDAP session. You call this function to unbind from the directory, close a connection, and discard the session handle. If the function succeeds, an **LDAP_SUCCESS** value is returned, and the connection between the client and server is broken down. If the function fails, it returns an error code.

Syntax

```
WINLDAPAPI ULONG LDAPAPI ldap_unbind_s(
    LDAP *ld
);
```

Parameters

ld

The session handle.

Example: Ldap_unbind_s

The following example uses `ldp.exe` and a network trace to illustrate use of the **Ldap_unbind_s** function.

To unbind and close the connection to the server

- In the Ldp window, on the **Connection** menu, click **Disconnect**.

The ldp window shows this information:

```
ldap_unbind(1d);
```

Looking at the network trace, frame 16 shows the Unbind request (**UnbindRequest**) for the client to the LDAP server. Frames 17 through 20 show the tear down of the TCP connection between the client and the server.

Frame 16

```
LDAP: ProtocolOp: UnbindRequest (2)
```

```
LDAP: MessageID // Note the Message ID in the Data = 0x0F
```

```
LDAP: ProtocolOp = UnbindRequest // Value in Data = 0x42
```

Frame 17

```
TCP: .A...F, len: 0, seq:2701669131-2701669131, ack:1171962498, win:17509, src: 389 dst: 1148
```

Frame 18

```
TCP: .A...., len: 0, seq:1171962498-1171962498, ack:2701669132, win:17188, src: 1148 dst: 389
```

Frame 19

```
TCP: .A...F, len: 0, seq:1171962498-1171962498, ack:2701669132, win:17188, src: 1148 dst: 389
```

Frame 20

```
TCP: .A...., len: 0, seq:2701669132-2701669132, ack:1171962499, win:17509, src: 389 dst: 1148
```

Interrogation

The interrogation APIs are used to query information that exists in the Directory Information Tree. These APIs may be called in several ways, for example, by a user working in the UI, by an OLE-DB application, or by an ADSI application.

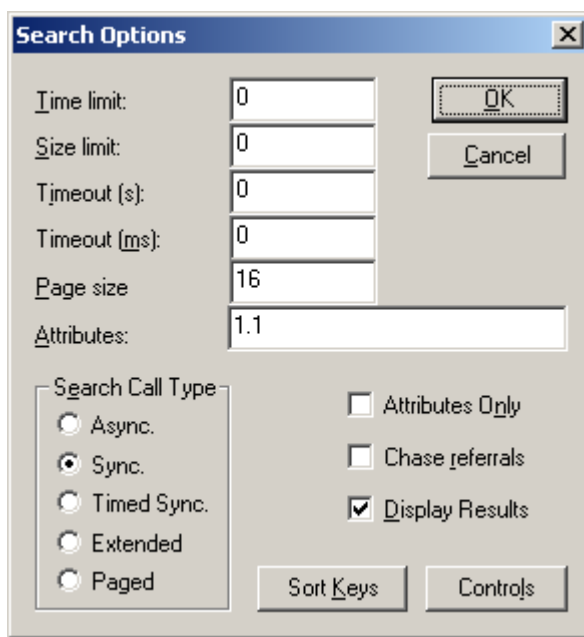
Interrogation Examples

To prepare for the interrogation examples, run the Demo.vbs script which will populate an OU in your domain called Demo. The Demo OU contains the entries and attributes needed so that you can follow the information in this paper. Next, bind to a DSA agent using the Ldp.exe LDAP client. See [“To bind to the DSA with the LDAP client,”](#) earlier in this paper.

After you have bound to the DSA, configure Ldp.exe as explained next.

To configure Ldp.exe Search options

1. In the Ldp window, on the **Options** menu, click **Search**.
2. In the **Search Options** dialog box specify the settings to use, as shown below:



Ldap_search_s()

The **Ldap_search_s()** function searches the LDAP directory, and returns a requested set of attributes for each entry matched. If the function succeeds, it returns the message ID of the search operation, and if it fails, it returns -1 and sets the session error parameters in the LDAP data structure. **Ldap_search_s()** begins a synchronous search operation whose scope is defined by the value of the *scope* parameter, described below.

Syntax

```
WINLDAPAPI ULONG LDAPAPI ldap_search(  
    LDAP *ld,  
    PCHAR base,  
    ULONG scope,  
    PCHAR filter,  
    PCHAR attrs[],  
    ULONG attrsonly,  
    LDAPMessage **res  
);
```

Parameters

ld

The session handle.

base

The distinguished name of the entry at which to start the search.

scope

Indicates the scope of the search, which is defined by the value of the scope parameter. The possible values for the scope parameter include:

- **LDAP_SCOPE_BASE**: Search the base entry only.
- **LDAP_SCOPE_ONELEVEL**: Search the base entry and all entries in the first level below the base.
- **LDAP_SCOPE_SUBTREE**: Search the base entry and all entries in the tree below the base.

filter

The search filter.

attrs

A null-terminated array of strings indicating the attributes to return for each matching entry. Pass NULL to retrieve all available attributes.

attrsonly

A boolean value that should be zero if both attribute types and values are to be returned, nonzero if only types are wanted.

res

Contains the results of the search upon completion of the call.

Comments

To control the characteristics of a session such as the time limits for the search or the result types, you can use two functions:

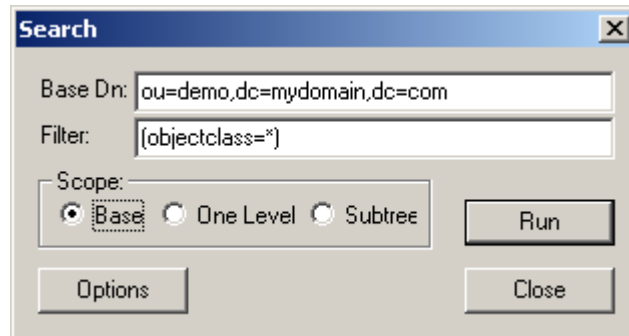
- **ldap_set_option** to set the value of the session parameters.
- **ldap_get_option** to access the current value of session-wide optional parameters.

Example: ldap_search_s

This example illustrates the use of the `ldap_search_s` function using `Ldp.exe`.

To set the scope of the Search operation to Base

1. In the `Ldp` window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, type `ou=demo,dc=mydomain,dc=com` in the **Base Dn** text box, type `(objectclass=*)` in the **Filter** text box, click **Base** in the **Scope** field, and click **Run**.



The `Ldp` window shows this information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 0, "(objectclass=*)",
attrList, 0, &msg)
Result <0>: (null)
Matched DNS:
Getting 1 entries:
>> Dn: ou=demo,dc=mydomain,dc=com
```

Important parameters include:

- **Base Object:** `ou=demo,dc=mydomain,dc=com`
- **Scope:** Base. Note that Base corresponds to the value 0.
- **Filter:** `(ObjectClass=*)`

Looking at the network trace, the search request (**SearchRequest**) is frame 18 and the search response (**SearchResponse**) is frame 19. Note that only one entry was returned for the base level search.

Frame 18

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: Scope = **Base Object**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: Filter Type = Present

LDAP: Attribute Type = objectclass

LDAP: Attribute Value = 1.1

Frame 19

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = ou=demo,dc=mydomain,dc=com

LDAP: MessageID

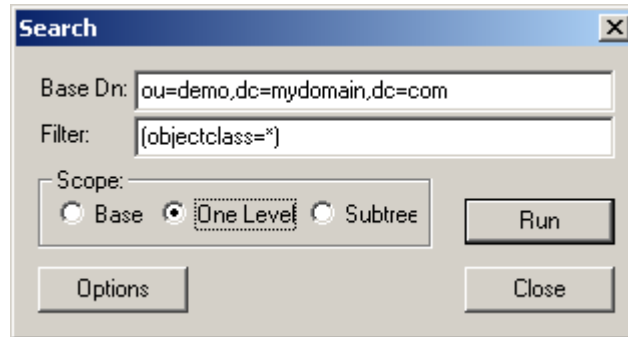
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To set the scope of the Search operation to One Level

1. In the Ldp window, on the **Browse** menu, click **Search**.

2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(objectclass=*)** in the **Filter** text box, and then click **Run**.



The Ldp window shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",
attrList, 0, &msg)
Result <0>: (null)
Matched DNs:
Getting 3 entries:
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level. Note that One Level corresponds to the value 1.
- **Filter:** (ObjectClass=*)

Looking at the network trace, the search request is frame 21 and the search response is frame 22. Note that only three entries were returned for the One Level search, and that the base DN was not included in the result set.

Frame 21

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: Filter Type = Present

LDAP: Attribute Type = objectclass

LDAP: Attribute Value = 1.1

Frame 22

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Support,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

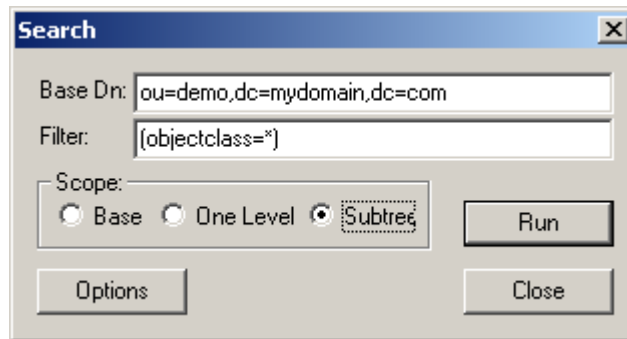
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To set the scope of the Search operation to Subtree

1. In the Ldp window, on the **Browse** menu, click **Search**.

- In the **Search** dialog box, click **Subtree** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(objectclass=*)** in the **Filter** text box, and then click **Run**.



The Ldp window shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 2, "(objectclass=*)",
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 20 entries:
```

```
>> Dn: OU=demo,DC=mydomain,DC=com
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Bob,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=John,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Pam,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Steve,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Barney,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Tim,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Dan,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Michael,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Lee,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Gary,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Alice,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Jessica,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Brent,OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Jim,OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Curtis,OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Mark,OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** Subtree. Note that Subtree corresponds to the value 2.
- **Filter:** ObjectClass=*)

Focusing on the network trace, the search request is frame 24 and the search response is frame 25. Note that 20 entries were returned for the Subtree search, and that the base DN was included in the result set.

Frame 24

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = whole Subtree**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: Filter Type = Present

LDAP: Attribute Type = objectclass

LDAP: Attribute Value = 1.1

Frame 25

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = CN=Bob,OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name =

CN=John,OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = CN=Pam,OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse
LDAP: Object Name =
CN=Steve,OU=Research,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name =
CN=Barney,OU=Research,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = CN=Tim,OU=Research,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = CN=Dan,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name =
CN=Micheal,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = CN=Lee,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = CN=Gary,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = CN=Alice,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name =
CN=Jessica,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name =
CN=Brent,OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: Object Name = CN=Jim,OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID

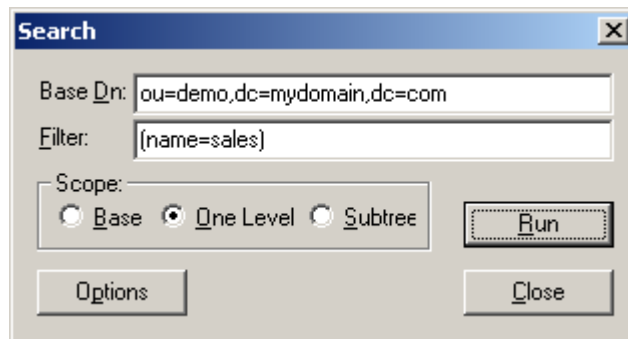
```

LDAP: ProtocolOp = SearchResponse
      LDAP: Object Name =
CN=Curtis,OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
      LDAP: Object Name = CN=Mark,OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse (simple)
      LDAP: Result Code = Success

```

To specify an Equality Match type Filter

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(name=sales)** in the **Filter** text box, and then click **Run**.
The filter type is *Equality Match*, *name* is the attribute type, and *sales* is the attribute value. This searches The Active Directory for an OU called *sales*.



The Ldp window shows the following information:

```

ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(name=sales)",
attrList, 0, &msg)
Result <0>: (null)
Matched DNS:
Getting 1 entries:
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com

```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level

-
- **Filter:** (name=sales)

Focusing on the network trace, the search request is frame 27 and the search response is frame 28. Note that only one entry was returned for the filtered search.

Frame 27

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type = Equality Match**

LDAP: **Attribute Type = name**

LDAP: **Attribute Value = sales**

LDAP: Attribute Value = 1.1

Frame 28

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

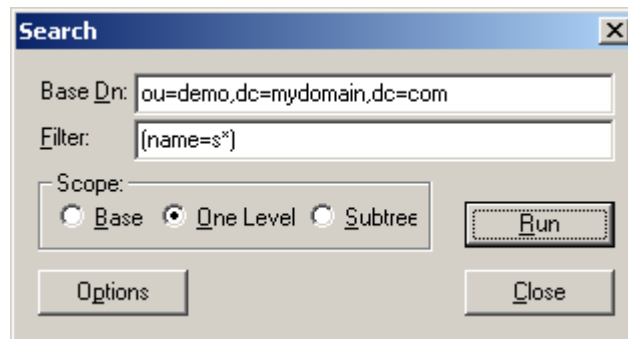
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To specify a simple substring type filter (<attr> = [<value>]*)

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(name=s*)** in the **Filter** text box, and then click **Run**.

The filter type is *Substrings*, the attribute type is *name*, and *s** is the substring value. This searches the Active Directory for all OU names that begin with the letter **s**.



The Ldp window shows this information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(name=s*)", attrList,
0, &msg)
Result <0>: (null)
Matched DNS:
Getting 2 entries:
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level
- **Filter:** (name=s*)

Looking at the network trace, the search request is frame 30 and the search response is frame 31. Note that two entries were returned for the filtered search.

Frame 30

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: Filter Type = **Substrings**

LDAP: **Attribute Type = name**

LDAP: **Substring (Initial) = s**

LDAP: Attribute Value = 1.1

Frame 31

LDAP: ProtocolOp: SearchResponse (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Support,OU=demo,DC=mydomain,DC=com

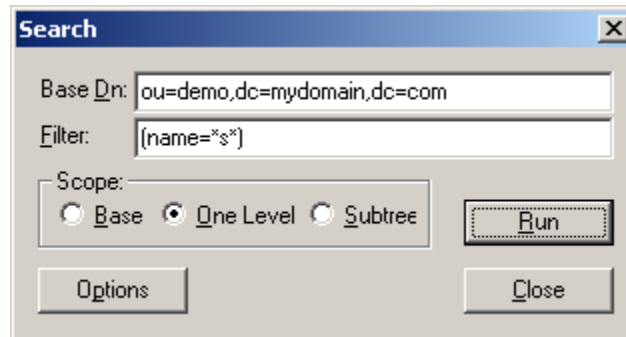
LDAP: MessageID

LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To specify filter of advanced substring type (<attr> = [<leading>] * [<value>] * [<trailing>])

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(name=*s*)** in the **Filter** text box, and then click **Run**.
The filter type is *Substrings*, the attribute type is *name*, and *s* is the substring value. This filter searches for all OU names that include the letter s in the name.



The Ldp window shows the following information:

```
Tdap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(name=*s*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 3 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level
- **Filter:** (name=*s*)

Looking at the network trace, the search request is frame 33 and the search response is frame 34. Note that three entries were returned for the filtered search.

Frame 33

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type = Substrings**

LDAP: **Attribute Type = name**

LDAP: **Substring (Any) = s**

LDAP: Attribute Value = 1.1

Frame 34

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Support,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

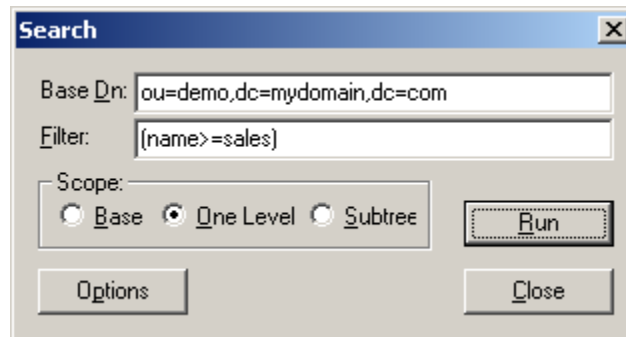
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To use greater than or equal (>=) as a filter type

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(name>=sales)** in the **Filter** text box, and then click **Run**.

The filter type is *greater or equal* (>=), the attribute type is *name*, and *sales* is the substring value. This filter searches for all OU names that are equal to or greater than the sales name.



The Ldp windows shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(name>=sales)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 2 entries:
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level
- **Filter:** (name=>sales)

Reviewing the network trace, the search request is frame 36 and the search response is frame 37. Note that two entries were returned for the filtered search.

Frame 36

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type = Greater Or Equal**

LDAP: **Attribute Type = name**

LDAP: **Attribute Value = sales**

LDAP: Attribute Value = 1.1

Frame 37

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Support,OU=demo,DC=mydomain,DC=com**

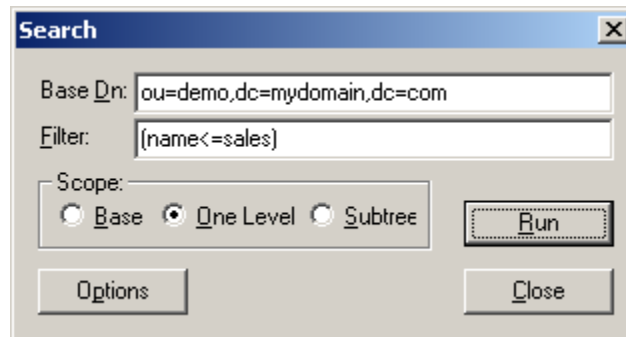
LDAP: MessageID

LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To use less than or equal (>=) as a filter type

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(name<=sales)** in the **Filter** text box, and then click **Run**.
The filter type is *less or equal* (<=), the attribute type is *name*, and *sales* is the substring value. This filter searches for all OU names that are equal to or less than the sales name.



The Ldp windows shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(name<=sales)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 2 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level
- **Filter:** (name<=sales)

Reviewing the network trace, the search request is frame 39 and the search response is frame 40. Note that two entries were returned for the filtered search.

Frame 39

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: Base Object = ou=demo,dc=mydomain,dc=com

LDAP: **Scope = Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type = Less Or Equal**

LDAP: **Attribute Type = name**

LDAP: **Attribute Value = sales**

LDAP: Attribute Value = 1.1

Frame 40

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

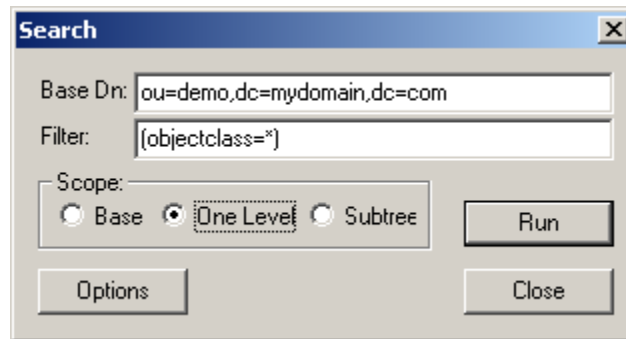
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To perform a search using filter - presence

<attr> = *

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(objectclass=*)** in the **Filter** text box, and then click **Run**.



The Ldp windows shows the following information:

```
Tdap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 3 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** One Level
- **Filter:** (objectclass=*)

Reviewing the network trace, the search request is frame 42 and the search response is frame 43. Note that three entries were returned for the filtered search.

Frame 42

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: **Base Object** = ou=demo,dc=mydomain,dc=com

LDAP: **Scope** = **Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type** = **Present**

LDAP: **Attribute Type** = **objectclass**

LDAP: Attribute Value = 1.1

Frame 43

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Research**,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Sales**,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Support**,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

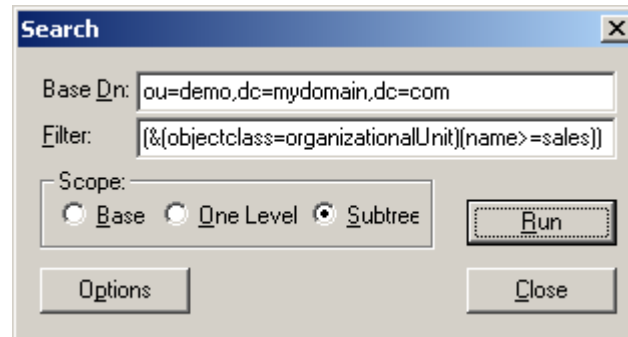
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To perform a complex search using filter AND

(&(<Filter>)(<Filter>)

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **Subtree** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(&(objectclass=organizationalUnit)(name>=sales))** in the **Filter** text box, and then click **Run**.



The Ldp windows shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 2,
"(&(objectclass=organizationalUnit)(name>=sales))", attrList, 0, &msg)
Result <0>: (null)
Matched DNS:
Getting 2 entries:
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important Parameters:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** Subtree
- **Filter:** (&(objectclass=organizationalUnit)(name>=sales))

Reviewing the network trace, the search request is frame 45 and the search response is frame 46. Note that three entries were returned for the filtered search.

Frame 45

```
LDAP: ProtocolOp: SearchRequest (3)
  LDAP: MessageID
  LDAP: ProtocolOp = SearchRequest
    LDAP: Base Object = ou=demo,dc=mydomain,dc=com
    LDAP: Scope = Whole Subtree
    LDAP: Deref Aliases = Never Deref Aliases
    LDAP: Size Limit = No Limit
    LDAP: Time Limit = No Limit
    LDAP: Attrs Only = 0 (0x0)
    LDAP: Filter Type = And
      LDAP: Filter Type = Equality Match
        LDAP: Attribute Type = objectclass
          LDAP: Attribute Value = organizationalUnit
        LDAP: Filter Type = Greater Or Equal
          LDAP: Attribute Type = name
            LDAP: Attribute Value = sales
          LDAP: Attribute Value = 1.1
```

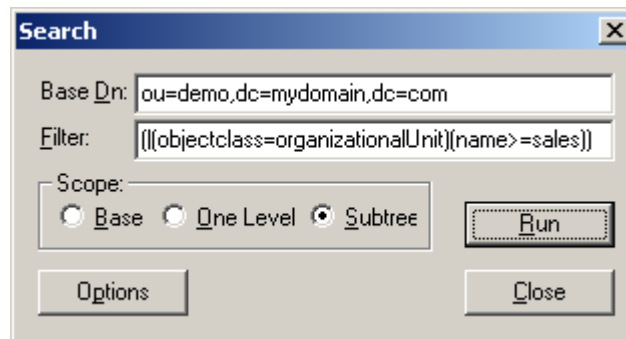
Frame 46

```
LDAP: ProtocolOp: SearchResponse (4)
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = OU=Support,OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse (simple)
    LDAP: Result Code = Success
```

To perform a complex search using filter OR

((<Filter>)(<Filter>))

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **Subtree** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **((objectclass=organizationalUnit)(name>=sales))** in the **Filter** text box, and then click **Run**.



The Ldp windows shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 2,
"(|(objectclass=organizationalUnit)(name>=sales))", attrList, 0, &msg)
```

Result <0>: (null)

Matched DNS:

Getting 6 entries:

```
>> Dn: OU=demo,DC=mydomain,DC=com
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Steve,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Tim,OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important Parameters:

- **Base Object:** -> ou=demo,dc=mydomain,dc=com
- **Scope:** -> Subtree
- **Filter:** ((objectclass=organizationalUnit)(name>=sales))

Reviewing the network trace, the search request is frame 48 and the search response is frame 49. Note that six entries were returned for the filtered search.

Frame 98

```
LDAP: ProtocolOp: SearchRequest (3)
  LDAP: MessageID
  LDAP: ProtocolOp = SearchRequest
    LDAP: Base Object = ou=demo,dc=mydomain,dc=com
    LDAP: Scope = Whole Subtree
    LDAP: Deref Aliases = Never Deref Aliases
    LDAP: Size Limit = No Limit
    LDAP: Time Limit = No Limit
    LDAP: Attrs Only = 0 (0x0)
    LDAP: Filter Type = Or
      LDAP: Filter Type = Equality Match
        LDAP: Attribute Type = objectclass
          LDAP: Attribute Value = organizationalUnit
        LDAP: Filter Type = Greater Or Equal
          LDAP: Attribute Type = name
            LDAP: Attribute Value = sales
          LDAP: Attribute Value = 1.1
```

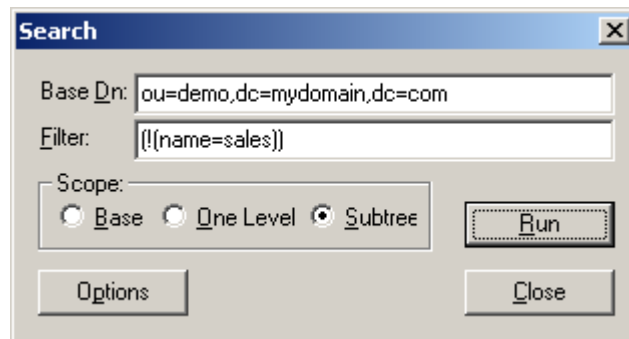
Frame 99

```
LDAP: ProtocolOp: SearchResponse (4)
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = OU=Research,OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = CN=Steve,OU=Research,OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = CN=Tim,OU=Research,OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
    LDAP: Object Name = OU=Sales,OU=demo,DC=mydomain,DC=com
  LDAP: MessageID
  LDAP: ProtocolOp = SearchResponse
```

LDAP: Object Name = OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: Protocol = SearchResponse (simple)
LDAP: Result Code = Success

To perform a complex search using filter - NOT (!(<Filter>))

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **Subtree** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **!(name=sales)** in the **Filter** text box, and then click **Run**.



The Ldp windows shows the following information:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 2, "!(name=sales)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNs:
```

```
Getting 19 entries:
```

```
>> Dn: OU=demo,DC=mydomain,DC=com  
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Bob,OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=John,OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Pam,OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Steve,OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Barney,OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Tim,OU=Research,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Dan,OU=Sales,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Michael,OU=Sales,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Lee,OU=Sales,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Gary,OU=Sales,OU=demo,DC=mydomain,DC=com  
>> Dn: CN=Alice,OU=Sales,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: CN=Jessica,OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Brent,OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Jim,OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Curtis,OU=Support,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Mark,OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameters include:

- **Base Object:** ou=demo,dc=mydomain,dc=com
- **Scope:** Subtree
- **Filter:** (!(name=sales))

Reviewing the network trace, the search request is frame 51 and the search response is frame 49. Note that nineteen entries were returned for the filtered search.

Frame 51

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: **Base Object** = **ou=demo,dc=mydomain,dc=com**

LDAP: **Scope** = **whole Subtree**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type** = **Not**

LDAP: **Filter Type** = **Equality Match**

LDAP: **Attribute Type** = **name**

LDAP: **Attribute Value** = **sales**

LDAP: Attribute Value = 1.1

Frame 52

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **CN=Bob,OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** =

CN=John,OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **CN=Pam,OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** =

CN=Steve,OU=Research,OU=demo,DC=mydomain,DC=com

LDAP: MessageID

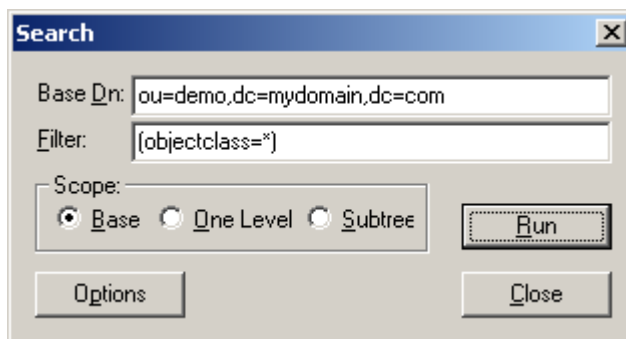
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name =**
CN=Barney,OU=Research,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Tim,OU=Research,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Dan,OU=Sales,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name =**
CN=Michael,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Lee,OU=Sales,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Gary,OU=Sales,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Alice,OU=Sales,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name =**
CN=Jessica,OU=Sales,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = OU=Support,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name =**
CN=Brent,OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Jim,OU=Support,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name =**
CN=Curtis,OU=Support,OU=demo,DC=mydomain,DC=com
LDAP: MessageID
LDAP: ProtocolOp = SearchResponse
LDAP: **Object Name = CN=Mark,OU=Support,OU=demo,DC=mydomain,DC=com**
LDAP: MessageID

LDAP: ProtocolOp = SearchResponse (simple)

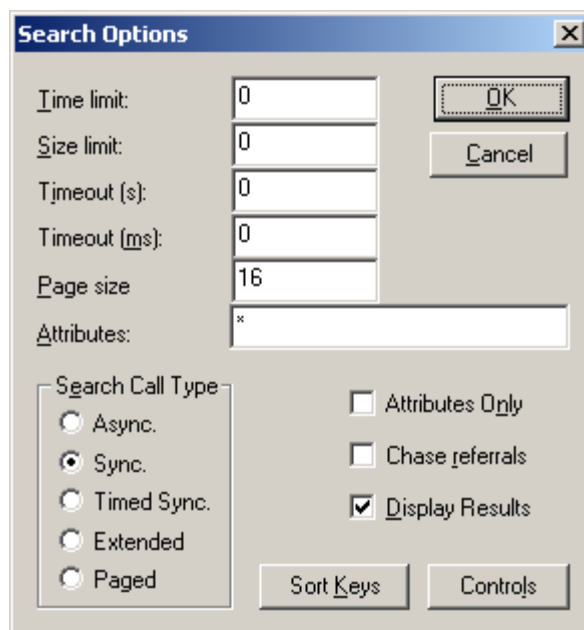
LDAP: Result Code = Success

To search for all returned attributes

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **Base** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(objectclass=*)** in the **Filter** text box, and then click **Run**.



3. In the Ldp window, on the **Options** menu, click **Search**.
4. In the **Search Options** dialog box specify the settings to use, and type * in the **Attributes** text box, as shown below:



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 0, "(objectclass=*)",
attrList, 0, &msg)
Result <0>: (null)
Matched DNS:
Getting 1 entries:
>> Dn: ou=demo,dc=mydomain,dc=com
    1> description: OU=Demo for Understanding LDAP;
    1> instanceType: 4;
    1> distinguishedName: OU=demo,DC=mydomain,DC=com;
    1> objectCategory: CN=Organizational-
Unit,CN=Schema,CN=Configuration,DC=mydomain,DC=com;
    2> objectClass: top; organizationalUnit;
    1> objectGUID: ?(K3?ny;
    1> ou: demo;
    1> name: demo;
    1> uSNChanged: 3040;
    1> uSNCreated: 3040;
    1> whenChanged: 19990828200325.0z;
    1> whenCreated: 19990828200325.0z;
```

Important parameter:

- **Attribute: = ***

Reviewing the network trace, the search request is frame 62 and the search response is frame 63. Note that one entries was returned with 12 attributes.

Frame 62

```
LDAP: ProtocolOp: SearchRequest (3)
LDAP: MessageID
LDAP: ProtocolOp = SearchRequest
LDAP: Base Object = ou=demo,dc=mydomain,dc=com
LDAP: Scope = Base Object
LDAP: Deref Aliases = Never Deref Aliases
LDAP: Size Limit = No Limit
LDAP: Time Limit = No Limit
LDAP: Attrs Only = 0 (0x0)
LDAP: Filter Type = Present
LDAP: Attribute Type = objectclass
LDAP: Attribute Value = *
```

Frame 63

LDAP: ProtocolOp: SearchResponse (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = ou=demo,dc=mydomain,dc=com

LDAP: Attribute Type = description

LDAP: Attribute Value = OU=Demo for Understanding LDAP

LDAP: Attribute Type = instanceType

LDAP: Attribute Value = 4

LDAP: Attribute Type = distinguishedName

LDAP: Attribute Value = OU=demo,DC=mydomain,DC=com

LDAP: Attribute Type = objectCategory

LDAP: Attribute Value = CN=Organizational-Unit,CN=Schema,CN=Configuration,DC=mydomain,DC=com

LDAP: Attribute Type = objectClass

LDAP: Attribute Value = top

LDAP: Attribute Value = organizationalUnit

LDAP: Attribute Type = objectGUID

LDAP: Attribute Value = ú\$ÿ |(K!+!úññÿ

LDAP: Attribute Type = ou

LDAP: Attribute Value = demo

LDAP: Attribute Type = name

LDAP: Attribute Value = demo

LDAP: Attribute Type = uSNChanged

LDAP: Attribute Value = 3040

LDAP: Attribute Type = uSNCreated

LDAP: Attribute Value = 3040

LDAP: Attribute Type = whenChanged

LDAP: Attribute Value = 19990828200325.0Z

LDAP: Attribute Type = whenCreated

LDAP: Attribute Value = 19990828200325.0Z

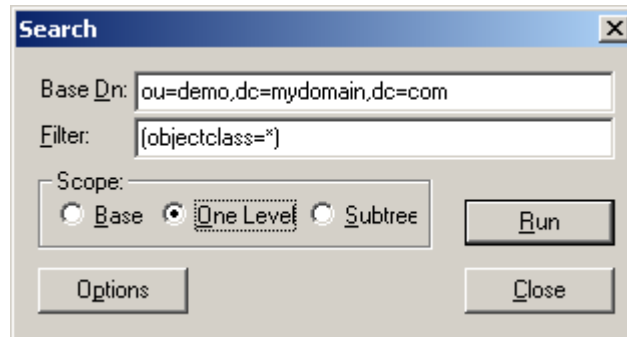
LDAP: MessageID

LDAP: ProtocolOp = SearchResponse (simple)

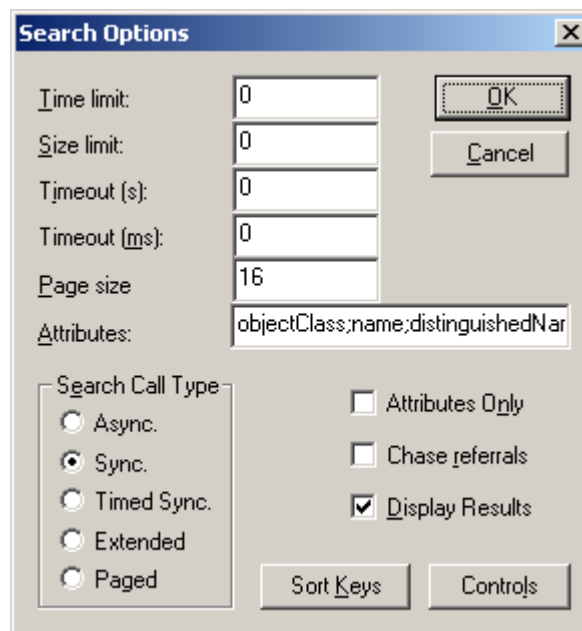
LDAP: Result Code = Success

To search for selected attributes Attributes – Selected

1. In the Ldp window, on the **Browse** menu, click **Search**.
2. In the **Search** dialog box, click **One Level** in the **Scope** field, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** text box, type **(objectclass=*)** in the **Filter** text box, and then click **Run**.



3. In the Ldp window, on the **Options** menu, click **Search**.
4. In the **Search Options** dialog box specify the settings to use, and type **objectClass;name;distinguishedName;canonicalName** in the **Attributes** text box, as shown below:



The Ldp window shows the following:

```
ldap_search_s(lld, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNs:
```

```
Getting 3 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
1> canonicalName: mydomain.com/demo/Research;
```

```
1> distinguishedName: OU=Research,OU=demo,DC=mydomain,DC=com;
```

```
2> objectClass: top; organizationalUnit;
```

```
1> name: Research;
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

```
1> canonicalName: mydomain.com/demo/Sales;
```

```
1> distinguishedName: OU=Sales,OU=demo,DC=mydomain,DC=com;
```

```
2> objectClass: top; organizationalUnit;
```

```
1> name: Sales;
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

```
1> canonicalName: mydomain.com/demo/Support;
```

```
1> distinguishedName: OU=Support,OU=demo,DC=mydomain,DC=com;
```

```
2> objectClass: top; organizationalUnit;
```

```
1> name: Support;
```

Important parameter:

- **Attributes:** canonicalName, distinguishedName, objectClass, name, UserPrincipalName

Reviewing the network trace, the search request is frame 66 and the search response is frame 67. Note that three entries were returned with 4 attributes. UserPrincipal name was not returned for any objects because it does not exist.

Frame 66

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: **Base Object** = **ou=demo,dc=mydomain,dc=com**

LDAP: **Scope** = **Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type** = **Present**

LDAP: **Attribute Type** = **objectclass**

LDAP: **Attribute Value** = **objectClass**

LDAP: **Attribute Value** = **name**

LDAP: **Attribute Value** = **distinguishedName**

LDAP: **Attribute Value** = **canonicalName**

LDAP: **Attribute Value** = **userPrincipalName**

Frame 67

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: **Attribute Type** = **canonicalName**

LDAP: **Attribute Value** = **mydomain.com/demo/Research**

LDAP: **Attribute Type** = **distinguishedName**

LDAP: **Attribute Value** = **OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: **Attribute Type** = **objectClass**

LDAP: **Attribute Value** = **top**

LDAP: **Attribute Value** = **organizationalUnit**

LDAP: **Attribute Type** = **name**

LDAP: **Attribute Value** = **Research**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Sales,OU=demo,DC=mydomain,DC=com**

LDAP: **Attribute Type** = **canonicalName**

LDAP: **Attribute Value** = **mydomain.com/demo/Sales**

LDAP: Attribute Type = distinguishedName
LDAP: Attribute Value = OU=Sales,OU=demo,DC=mydomain,DC=com

LDAP: Attribute Type = objectClass
LDAP: Attribute Value = top
LDAP: Attribute Value = organizationalUnit

LDAP: Attribute Type = name
LDAP: Attribute Value = Sales

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: Object Name = OU=Support,OU=demo,DC=mydomain,DC=com

LDAP: Attribute Type = canonicalName
LDAP: Attribute Value = mydomain.com/demo/Support

LDAP: Attribute Type = distinguishedName
LDAP: Attribute Value = OU=Support,OU=demo,DC=mydomain,DC=com

LDAP: Attribute Type = objectClass
LDAP: Attribute Value = top
LDAP: Attribute Value = organizationalUnit

LDAP: Attribute Type = name
LDAP: Attribute Value = Support

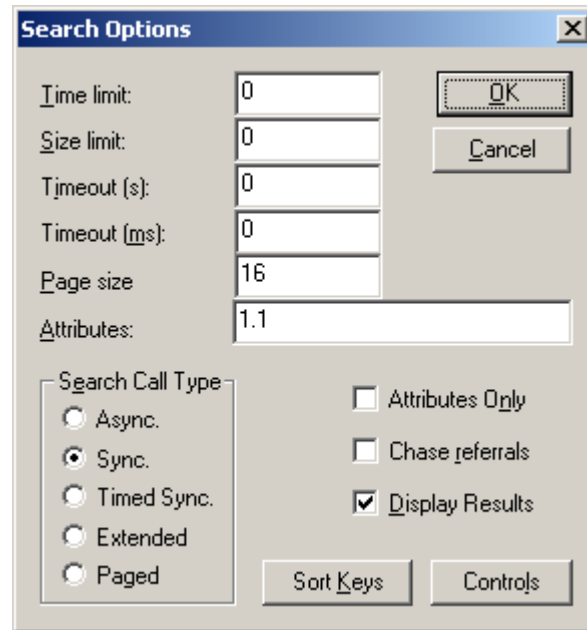
LDAP: MessageID

LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To set attributes to OID 1.1 (to prevent attributes from being returned)

1. In the Ldp window, on the **Options** menu, click **Search**.
2. In the **Search Options** dialog box specify the settings to use, and type **1.1** in the **Attributes** text box, as shown below:



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 3 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Important parameter:

- **Attributes:** set to **1.1**

Reviewing the network trace, the search request is frame 69 and the search response is frame 70. Note that three entries were returned with no attributes. The OID 1.1 does not equate to any defined attributes; therefore, none are returned.

Frame 69

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: **Base Object** = **ou=demo,dc=mydomain,dc=com**

LDAP: **Scope** = **Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type** = **Present**

LDAP: **Attribute Type** = **objectclass**

LDAP: **Attribute Value** = **1.1**

Frame 70

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Sales,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Support,OU=demo,DC=mydomain,DC=com**

LDAP: MessageID

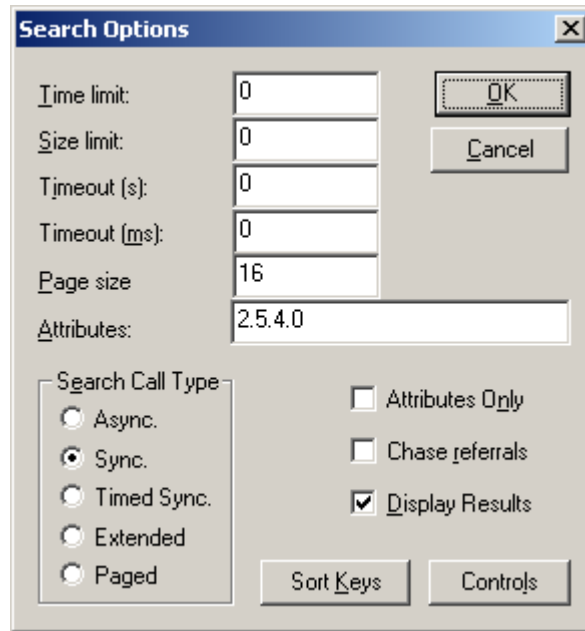
LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

To set attributes to OID 2.5.4.0 (objectClass)

1. In the Ldp window, on the **Options** menu, click **Search**.

2. In the **Search Options** dialog box specify the settings to use, and type **2.5.4.0** in the **Attributes** text box, as shown below:



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 3 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com  
    2> objectClass: top; organizationalUnit;  
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com  
    2> objectClass: top; organizationalUnit;  
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com  
    2> objectClass: top; organizationalUnit;
```

Important parameter:

- **Attributes:** set to **2.5.4.0**

Reviewing the network trace, the search request is frame 72 and the search response is frame 73. Note that three entries were returned with one attribute. The OID 2.5.4.0 equates to objectclass.

Frame 72

LDAP: ProtocolOp: **SearchRequest** (3)

LDAP: MessageID

LDAP: ProtocolOp = SearchRequest

LDAP: **Base Object** = **ou=demo,dc=mydomain,dc=com**

LDAP: **Scope** = **Single Level**

LDAP: Deref Aliases = Never Deref Aliases

LDAP: Size Limit = No Limit

LDAP: Time Limit = No Limit

LDAP: Attrs Only = 0 (0x0)

LDAP: **Filter Type** = **Present**

LDAP: **Attribute Type** = **objectclass**

LDAP: **Attribute Value** = **2.5.4.0**

Frame 73

LDAP: ProtocolOp: **SearchResponse** (4)

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Research,OU=demo,DC=mydomain,DC=com**

LDAP: **Attribute Type** = **objectClass**

LDAP: **Attribute Value** = **top**

LDAP: **Attribute Value** = **organizationalUnit**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Sales,OU=demo,DC=mydomain,DC=com**

LDAP: **Attribute Type** = **objectClass**

LDAP: **Attribute Value** = **top**

LDAP: **Attribute Value** = **organizationalUnit**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse

LDAP: **Object Name** = **OU=Support,OU=demo,DC=mydomain,DC=com**

LDAP: **Attribute Type** = **objectClass**

LDAP: **Attribute Value** = **top**

LDAP: **Attribute Value** = **organizationalUnit**

LDAP: MessageID

LDAP: ProtocolOp = SearchResponse (simple)

LDAP: Result Code = Success

Ldap_compare_s()

The **ldap_compare_s** function is used to determine whether an attribute for a given entry holds a known value.

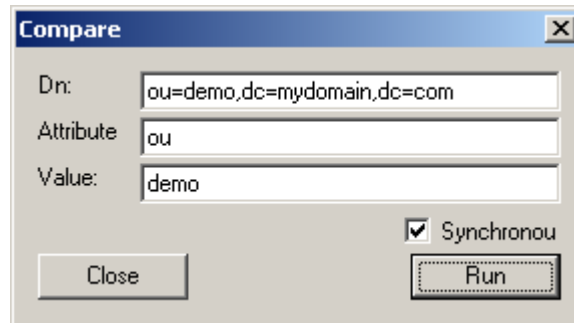
If the function succeeds, and the attribute and known values match, the return value is **LDAP_COMPARE_TRUE**. If the values do not match, the return value is **LDAP_COMPARE_FALSE**. If the function fails, it returns an error code.

Syntax

```
#include <winldap.h> WINLDAPAPI ULONG LDAPAPI ldap_compare_s(  
    LDAP *ld,  
    PCHAR dn,  
    PCHAR attr,  
    CHAR value  
);
```

To use the Compare function to search for the Demo OU (returns True)

1. In the Ldp window, click **Compare** on the **Browse** menu.
2. In the **Dn** text box, type **ou=demo,dc=mydomain,dc=com**, type **ou** in the **Attribute** box, type **demo** in the **Value** box, and then click **Run**.



The Ldp window shows the following:

```
ldap_compare_s(0x712040, "ou=demo,dc=mydomain,dc=com", "ou", "demo")
Results: TRUE. <6>
```

Reviewing the network trace, the compare request is frame 75 and the compare response is frame 76. Frame 76 states that the comparison was true.

Frame 75

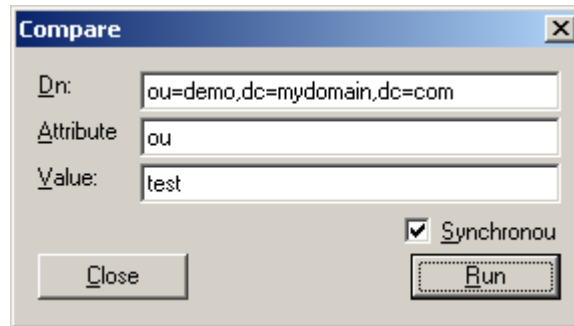
```
LDAP: ProtocolOp: CompareRequest (14)
  LDAP: MessageID
  LDAP: ProtocolOp = CompareRequest
    LDAP: Object Name = ou=demo,dc=mydomain,dc=com
    LDAP: Attribute Type = ou
      LDAP: Attribute Value = demo
```

Frame 76

```
LDAP: ProtocolOp: CompareResponse (15)
  LDAP: MessageID
  LDAP: ProtocolOp = CompareResponse
    LDAP: Result Code = Compare True
```

To use the Compare function to search for an OU called Test (returns False)

1. In the Ldp window, click **Compare** on the **Browse** menu.
2. In the **Dn** text box, type **ou=demo,dc=mydomain,dc=com**, type **ou** in the **Attribute** box, type **test** in the **Value** box, and then click **Run**.



The Ldp window shows the following:

```
ldap_compare_s(0x712040, "ou=demo,dc=mydomain,dc=com", "ou", "test")  
Results: FALSE. <5>
```

Reviewing the network trace, the compare request is frame 78 and the compare response is frame 79. Frame 79 states that the comparison was false.

Frame 78

```
LDAP: ProtocolOp: CompareRequest (14)  
LDAP: MessageID  
LDAP: ProtocolOp = CompareRequest  
LDAP: Object Name = ou=demo,dc=mydomain,dc=com  
LDAP: Attribute Type = ou  
LDAP: Attribute Value = test
```

Frame 79

```
LDAP: ProtocolOp: CompareResponse (15)  
LDAP: MessageID  
LDAP: ProtocolOp = CompareResponse  
LDAP: Result Code = Compare False
```

Update

The Update APIs allow the client to modify information that exists in the directory information tree. The client may either create new entries or modify and delete existing entries.

Ldap_add_s()

The **ldap_add_s** function adds an entry to a directory information tree. This function returns LDAP_SUCCESS if it succeeds, and returns an error code if it fails.

Syntax

```
#include <winldap.h> WINLDAPAPI ULONG LDAPAPI ldap_add_s(  
    LDAP *ld,  
    PCHAR dn,  
    LDAPMod *attrs[]  
);
```

Parameters

ld

The session handle.

dn

The distinguished name of the entry to add.

attrs

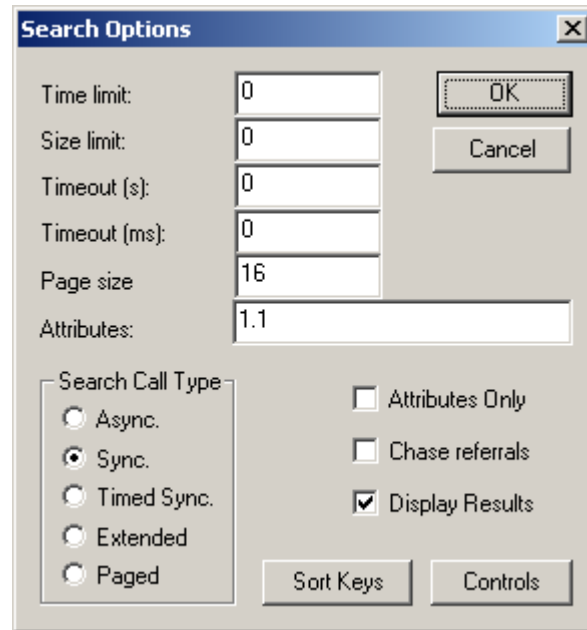
An array of pointers to **LDAPMod** structures that hold information required to perform a modification. Each structure specifies a single attribute.

For the **ldap_add_s** function to be successful, the following conditions must be met:

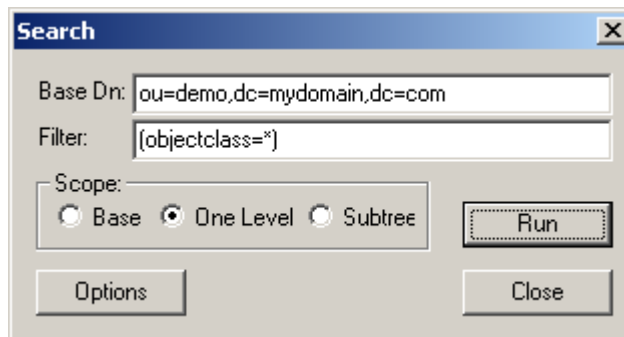
- The parent object must exist as a container
- The new DN must be unique within the directory information tree
- The new object must conform to the schema
- Access control permissions must permit the creation of the object

To perform a test search of the Demo OU

1. In the Ldp window, click **Search** on the **Options** menu.
2. In the **Search Options** dialog box, type **1.1** in the **Attributes** box and set the options as shown below, and then click **OK**.



3. Click **Search** on the **Browse** menu, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(objectclass=*)** in the **Filter** box, select **One Level** in the **Scope** field, and then click **Run**.



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)  
Result <0>: (null)
```

Matched DNS:

Getting 3 entries:

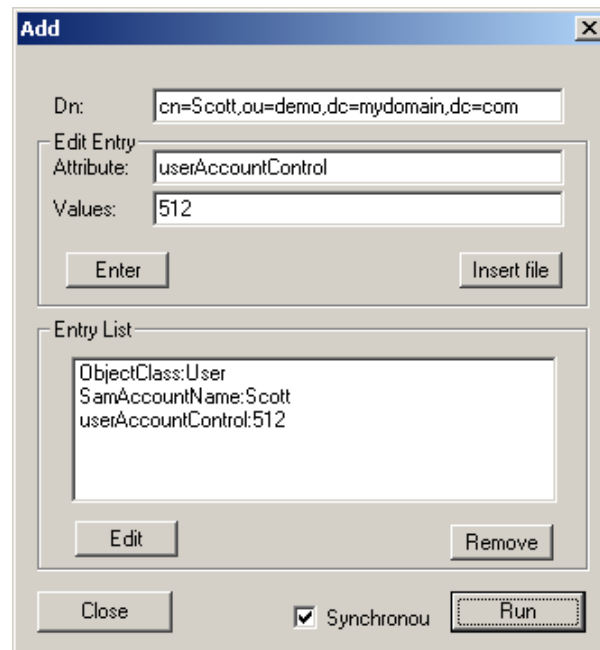
>> Dn: **OU=Research**,OU=demo,DC=mydomain,DC=com

>> Dn: **OU=Sales**,OU=demo,DC=mydomain,DC=com

>> Dn: **OU=Support**,OU=demo,DC=mydomain,DC=com

To add an entry to the directory tree (Update using `ldap_add_s` function)

1. In the Ldp window, click **Add** on the **Browse** menu.
2. In the **Add** dialog box, type **cn=Scott,ou=demo,dc=mydomain,dc=com** in the **Dn** box, type **userAccountControl** in the **Attribute** box, type **512** in the **Values** box, click **Enter**, and then click **Run**.



The Ldp window shows the following:

***Calling Add...

Added {cn=Scott,ou=demo,dc=mydomain,dc=com}

This configuration will add a new user named Scott. The schema states that one must include the **SamAccountName** and **objectClass** attributes. The **userAccountControl** attribute is set to enable the user account. Network frames 30 and 31 describe this process.

Frame 30

LDAP: ProtocolOp: **AddRequest** (8)

LDAP: MessageID

LDAP: ProtocolOp = AddRequest

LDAP: **Object Name** = cn=Scott,ou=Demo,dc=mydomain,dc=com

LDAP: **Attribute Type** = ObjectClass

LDAP: **Attribute Value** = User

LDAP: **Attribute Type** = SamAccountName

LDAP: **Attribute Value** = Scott

LDAP: **Attribute Type** = UserAccountControl

LDAP: **Attribute Value** = 512

Frame 31

LDAP: ProtocolOp: **AddResponse** (9)

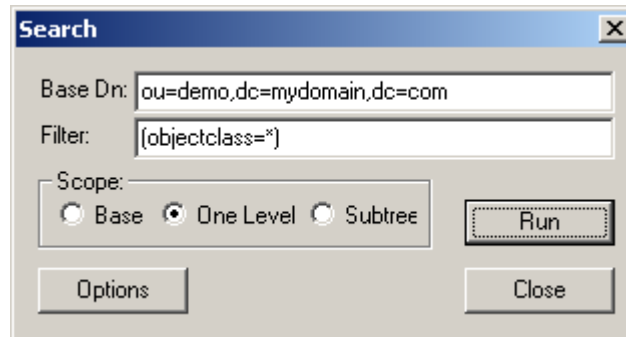
LDAP: MessageID

LDAP: ProtocolOp = AddResponse

LDAP: **Result Code** = Success

To confirm that the preceding update was successful

1. In the Ldp window, click **Search** on the **Browse** menu.
2. In the **Search** dialog box, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(objectclass=*)** in the **Filter** box, select **One Level** in the **Scope** field, and then click **Run**.



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 4 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: CN=Scott,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

The **ldap_add_s** function call was successful and user Scott appears as a new entry.

Ldap_modify_s()

The **ldap_modify_s** function changes an existing entry; this includes adding, deleting, and modifying attributes. This function returns LDAP_SUCCESS if it succeeds, and returns an error code if it fails.

Syntax

```
WINLDAPAPI ULONG LDAPAPI ldap_modify_s(  
    LDAP *ld,  
    PCHAR dn,  
    LDAPModA *mods[]  
);
```

Parameters

ld

The session handle.

dn

The name of the entry to modify.

mods

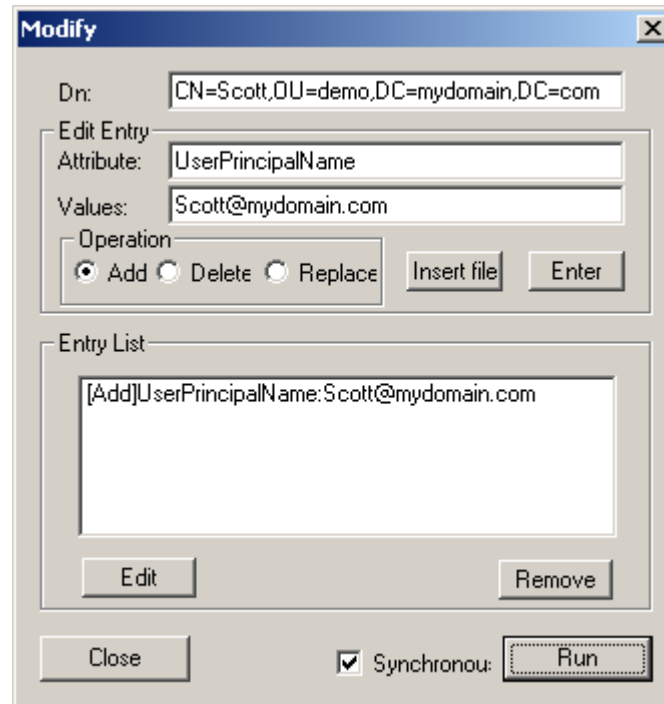
A null-terminated array of modifications to make to the entry.

For the **ldap_modify_s** function to be successful, the following conditions must be met:

- The object must exist
- All attribute modifications must succeed
- The resulting entry must conform to the schema
- Access controls must permit the modification of the entry

To modify an entry by adding an attribute

1. In the Ldp window, click **Modify** on the **Browse** menu.
2. In the **Modify** dialog box, type **cn=Scott,ou=demo,dc=mydomain,dc=com** in the **Dn** box, type **userPrincipalName** in the **Attribute** box, type **Scott@mydomain.com** in the **Values** box, click **Add**, and then click **Run**.



The Ldp window shows the following:

```
***Call Modify...
```

```
Modified "CN=Scott,OU=demo,DC=mydomain,DC=com"
```

This configuration adds a new attribute to the entry named Scott. Frames 36 and 37 from the network trace demonstrate the modify request and response.

Frame 36

LDAP: ProtocolOp: **ModifyRequest** (6)

LDAP: MessageID

LDAP: ProtocolOp = ModifyRequest

LDAP: **Object Name** = CN=Scott,OU=demo,DC=mydomain,DC=com

LDAP: **Operation** = **Add**

LDAP: **Attribute Type** = **UserPrincipalName**

LDAP: **Attribute Value** = **Scott@mydomain.com**

Frame 37

LDAP: ProtocolOp: **ModifyResponse** (7)

LDAP: MessageID

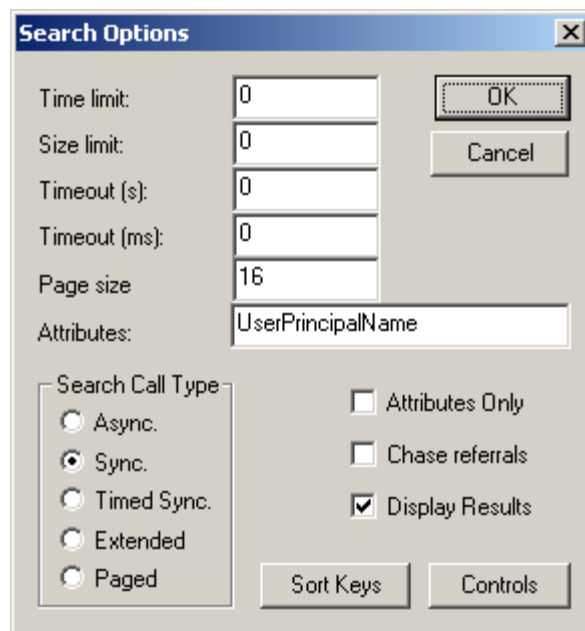
LDAP: ProtocolOp = ModifyResponse

LDAP: **Result Code** = **Success**

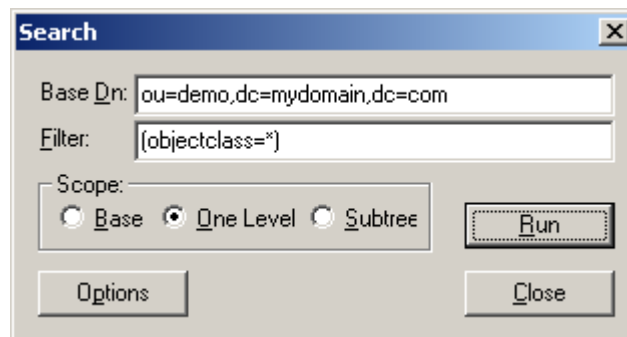
To confirm that the preceding Modify operation was successful

1. In the Ldp window, click **Search** on the **Options** menu.

2. In the **Search Options** dialog box, type **UserPrincipalName** in the **Attributes** box, and then click **OK**.



3. Click **Search** on the **Browse** menu.
4. In the **Search** dialog box, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(objectclass=*)** in the **Filter** box, select **One Level** in the **Scope** field, and then click **Run**.



The Ldp window shows the following:

```
ldap_search_s(lld, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNs:
```

```
Getting 4 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

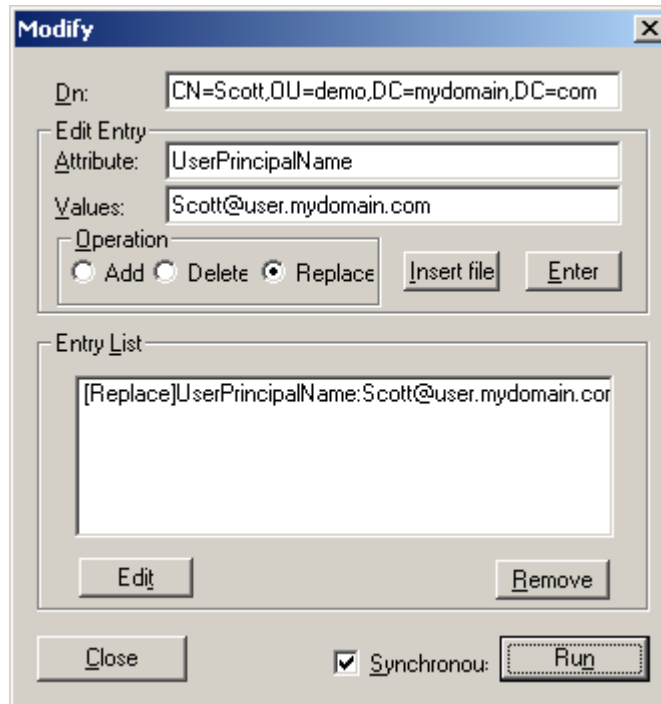
```
>> Dn: CN=Scott,OU=demo,DC=mydomain,DC=com
```

```
    1> userPrincipalName: Scott@mydomain.com;
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

To modify an entry by replacing an attribute

1. In the Ldp window, click **Modify** on the **Browse** menu.
2. In the **Modify** dialog box, type **cn=Scott,ou=demo,dc=mydomain,dc=com** in the **Dn** box, type **userPrincipalName** in the **Attribute** box, type **Scott@user.mydomain.com** in the **Values** box, click **Replace**, and then click **Run**.



The Ldp window shows the following:

***Call Modify...

Modified "CN=Scott,OU=demo,DC=mydomain,DC=com".

This configuration replaces an attribute to the entry named Scott. Frames 42 and 43 from the network trace demonstrate the modify request and response.

Frame 42

LDAP: Protocolop: **ModifyRequest** (6)

LDAP: MessageID

LDAP: Protocolop = ModifyRequest

LDAP: **Object Name** = CN=Scott,OU=demo,DC=mydomain,DC=com

LDAP: **operation** = **Replace**

LDAP: **Attribute Type** = **UserPrincipalName**

LDAP: **Attribute Value** = **Scott@user.mydomain.com**

Frame 43

LDAP: ProtocolOp: **ModifyResponse** (7)

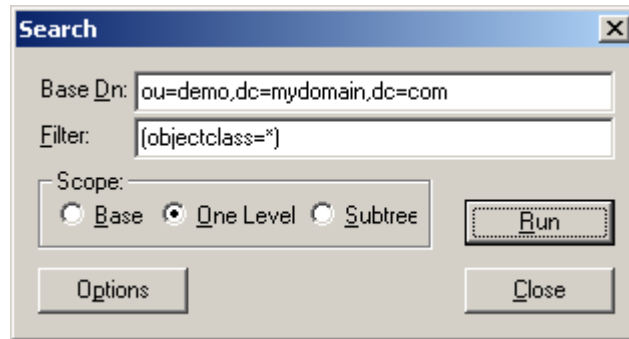
LDAP: MessageID

LDAP: ProtocolOp = ModifyResponse

LDAP: **Result Code = Success**

To confirm that the preceding replace operation was successful

1. In the Ldp window, click **Search** on the **Browse** menu.
2. In the **Search** dialog box, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(objectclass=*)** in the **Filter** box, select **One Level** in the **Scope** field, and then click **Run**.



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 4 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
```

```
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
```

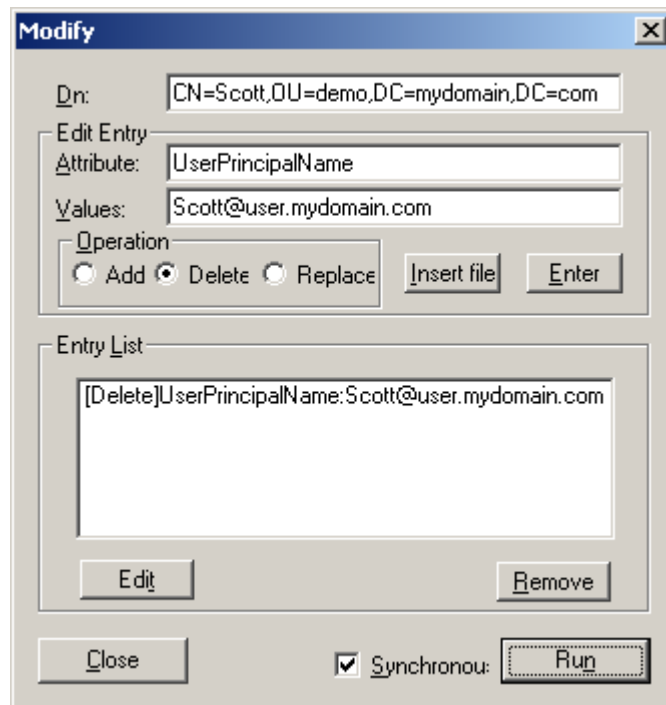
```
>> Dn: CN=Scott,OU=demo,DC=mydomain,DC=com
```

```
1> userPrincipalName: scott@user.mydomain.com;
```

```
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

To modify an entry by deleting an attribute

3. In the Ldp window, click **Modify** on the **Browse** menu.
4. In the **Modify** dialog box, type **cn=Scott,ou=demo,dc=mydomain,dc=com** in the **Dn** box, type **userPrincipalName** in the **Attribute** box, type **Scott@user.mydomain.com** in the **Values** box, click **Delete**, and then click **Run**.



The Ldp window shows the following:

***Call Modify...

Modified "CN=Scott,OU=demo,DC=mydomain,DC=com".

This configuration deletes an attribute on the entry named Scott. Frames 48 and 49 from the network trace demonstrate the modify request and response.

Frame 48

LDAP: ProtocolOp: **ModifyRequest** (6)

LDAP: MessageID

LDAP: ProtocolOp = ModifyRequest

LDAP: **Object Name** = CN=Scott,OU=demo,DC=mydomain,DC=com

LDAP: **Operation** = **Delete**

LDAP: **Attribute Type** = **UserPrincipalName**

LDAP: **Attribute Value** = **Scott@user.mydomain.com**

Frame 49

LDAP: ProtocolOp: **ModifyResponse** (7)

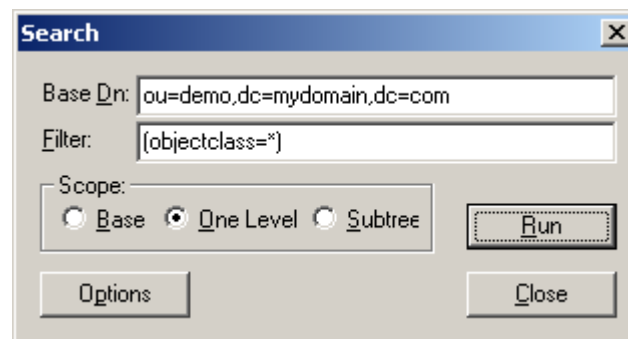
LDAP: MessageID

LDAP: ProtocolOp = ModifyResponse

LDAP: **Result Code** = **Success**

To confirm that preceding delete operation was successful

5. In the Ldp window, click **Search** on the **Browse** menu.
6. In the **Search** dialog box, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(objectclass=*)** in the **Filter** box, select **One Level** in the **Scope** field, and then click **Run**.



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 1, "(objectclass=*)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNS:
```

```
Getting 4 entries:
```

```
>> Dn: OU=Research,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Sales,OU=demo,DC=mydomain,DC=com
>> Dn: CN=Scott,OU=demo,DC=mydomain,DC=com
>> Dn: OU=Support,OU=demo,DC=mydomain,DC=com
```

Ldap_modrdn2_s()

The `ldap_modrdn_s` function changes the relative distinguished name of an LDAP entry. This function returns **LDAP_SUCCESS** if it succeeds, and returns an error code if it fails.

Syntax

```
WINLDAPAPI ULONG LDAPAPI ldap_modrdn_s (
    LDAP *ExternalHandle,
    PCHAR DistinguishedName,
    PCHAR NewDistinguishedName
    ..INT DeleteOldRDN
);
```

Parameters

ExternalHandle

The session handle.

DistinguishedName

The distinguished name of the entry to be changed.

NewDistinguishedName

The new relative distinguished name to give the entry. This is an OUT parameter.

DeleteOldRDN

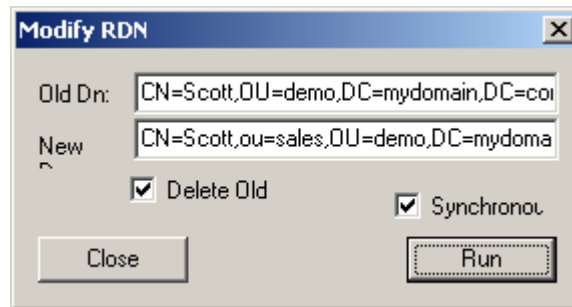
Set to TRUE to delete the old relative distinguished name and set to FALSE to retain it.

For the `ldap_modrdn2_s` function to be successful, the following conditions must be met:

- The object must exist
- The new RDN must be unique
- Access control permissions must allow for the modification of the entry

To change the RDN of an entry

7. In the Ldp window, click **Modify RDN** on the **Browse** menu.
8. In the **Modify RDN** dialog box, type **cn=Scott,ou=demo,dc=mydomain,dc=com** in the **Old Dn** box, type **cn=Scott,ou=sales,ou=demo,dc=mydomain,dc=com** in the **New DN** box, click **Delete Old**, and then click **Run**.



The Ldp window shows the following:

***Calling ModifyRdn2...

Rdn "CN=Scott,OU=demo,DC=mydomain,DC=com" modified to "CN=Scott,ou=sales,OU=demo,DC=mydomain,DC=com"

This configuration modifies the RDN of a user named Scott, thus moving him to a another container (the sales OU). Network frames 54 and 55 illustrate this process.

Frame 54

LDAP: ProtocolOp: **ModifyDNRequest** (12)

LDAP: MessageID

LDAP: ProtocolOp = ModifyDNRequest

LDAP: **Object Name** = CN=Scott,OU=demo,DC=mydomain,DC=com

LDAP: **New RDN** = CN=Scott

LDAP: **Delete Old RDN** = 255 (0xFF)

LDAP: **New Superior** = OU=Sales,OU=demo,DC=mydomain,DC=com

Frame 55

LDAP: ProtocolOp: **ModifyDNResponse** (13)

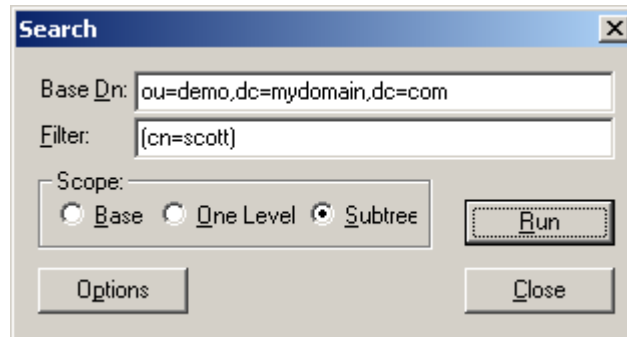
LDAP: MessageID

LDAP: ProtocolOp = ModifyDNResponse

LDAP: **Result Code** = Success

To confirm that preceding RDN change was successful

1. In the Ldp window, click **Search** on the **Browse** menu.
2. In the **Search** dialog box, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(cn=scott)** in the **Filter** box, select **Subtree** in the **Scope** field, and then click **Run**.



The Ldp window shows the following:

```
ldap_search_s(1d, "ou=demo,dc=mydomain,dc=com", 2, "(cn=scott)",
attrList, 0, &msg)
Result <0>: (null)
Matched DNS:
Getting 1 entries:
>> Dn: CN=Scott,OU=Sales,OU=demo,DC=mydomain,DC=com
```

The **ldap_modrdn2_s** function call succeeds, and Scott appears as a new entry entry in the Sales OU container.

Ldap_delete_s()

The **ldap_delete_s** function removes a leaf entry from the directory tree; this is a synchronous operation. If the function succeeds, it returns **LDAP_SUCCESS**; if it fails, it returns an error code.

Syntax

```
#include <winldap.h> WINLDAPAPI ULONG LDAPAPI ldap_delete_s(  
    LDAP *ld,  
    PCHAR dn  
);
```

Parameters

ld

The session handle.

dn

The distinguished name of the entry to delete.

For the **ldap_modrdn2_s** function to succeed, the following conditions must be met:

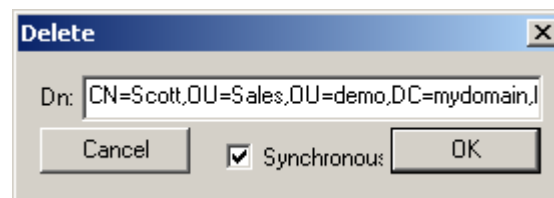
- The object must exist
- The object must not have children.
- Access controls must permit the deletion of the entry

To delete a leaf entry from the directory tree

9. In the Ldp window, click **Delete** on the **Browse** menu.

10. In the **Delete** dialog box, type

cn=Scott,ou=sales,ou=demo,dc=mydomain,dc=com in the **Dn** box, and



click **OK**.

The Ldp window shows the following information:

```
ldap_delete_s(1d, "CN=Scott,OU=Sales,OU=demo,DC=mydomain,DC=com");  
Deleted "CN=Scott,OU=Sales,OU=demo,DC=mydomain,DC=com"
```

This configuration deletes of a user entry named Scott. Network frames 60 and 61 illustrate this process.

Frame 60

LDAP: Protocolop: **DelRequest** (10)

LDAP: MessageID

LDAP: Protocolop = **DelRequest**

LDAP: Object Name = CN=Scott,OU=Sales,OU=demo,DC=mydomain,DC=com

Frame 61

LDAP: Protocolop: **DelResponse** (11)

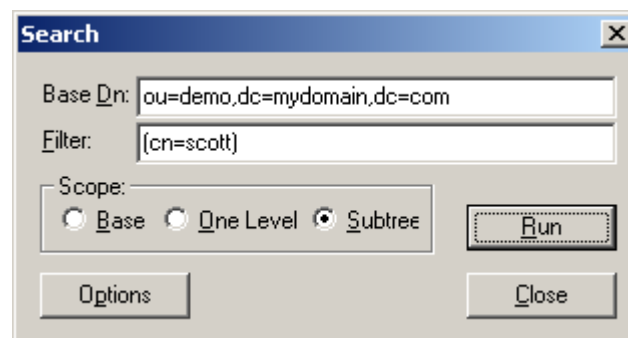
LDAP: MessageID

LDAP: Protocolop = **DelResponse**

LDAP: Result Code = Success

To confirm that the preceding delete operation was successful

1. In the Ldp window, click **Search** on the **Browse** menu.
2. In the **Search** dialog box, type **ou=demo,dc=mydomain,dc=com** in the **Base Dn** box, type **(cn=scott)** in the **Filter** box, select **Subtree** in the **Scope** field, and then click **Run**.



The Ldp window shows the following information:

```
ldap_search_s(ld, "ou=demo,dc=mydomain,dc=com", 2, "(cn=scott)",  
attrList, 0, &msg)
```

```
Result <0>: (null)
```

```
Matched DNs:
```

```
Getting 0 entries:
```

The **ldap_modrdn2_s** succeeded and a user named Scott does not exist as an entry in the Sales OU container.

Interpreting LDAP Errors

This section lists the LDAP error codes and discusses the use of Network Monitor traces to help interpret LDAP errors.

LDAP Error Codes

The following table lists the error codes encountered in LDAP applications.

Code	Value	Description
LDAP_SUCCESS	0x00	The call completed successfully.
LDAP_OPERATIONS_ERROR	0x01	Operations error occurred.
LDAP_PROTOCOL_ERROR	0x02	Protocol error occurred.
LDAP_TIMELIMIT_EXCEEDED	0x03	Time limit has exceeded.
LDAP_SIZELIMIT_EXCEEDED	0x04	Size limit has exceeded.
LDAP_COMPARE_FALSE	0x05	Compare yielded FALSE.
LDAP_COMPARE_TRUE	0x06	Compare yielded TRUE.
LDAP_AUTH_METHOD_NOT_SUPPORTED	0x07	The authentication method is not supported.
LDAP_STRONG_AUTH_REQUIRED	0x08	Strong authentication is required.
LDAP_REFERRAL_V2	0x09	LDAP v. 2 referral.
LDAP_PARTIAL_RESULTS	0x09	Partial results and referrals received.
LDAP_REFERRAL	0x0a	Referral occurred.
LDAP_ADMIN_LIMIT_EXCEEDED	0x0b	Administration limit on the server has exceeded.
LDAP_UNAVAILABLE_CRIT_EXTENSION	0x0c	Critical extension is unavailable.
LDAP_CONFIDENTIALITY_REQUIRED	0x0d	Confidentiality is required.
LDAP_NO_SUCH_ATTRIBUTE	0x10	Requested attribute does not exist.
LDAP_UNDEFINED_TYPE	0x11	The type is not defined.
LDAP_INAPPROPRIATE_MATCHING	0x12	An inappropriate matching occurred.
LDAP_CONSTRAINT_VIOLATION	0x13	A constraint violation occurred.
LDAP_ATTRIBUTE_OR_VALUE_EXISTS	0x14	The attribute exists or the value has been assigned.
LDAP_INVALID_SYNTAX	0x15	The syntax is invalid.
LDAP_NO_SUCH_OBJECT	0x20	Object does not exist.
LDAP_ALIAS_PROBLEM	0x21	The alias is invalid.
LDAP_INVALID_DN_SYNTAX	0x22	The distinguished name has an invalid syntax.
LDAP_IS_LEAF	0x23	The object is a leaf.
LDAP_ALIAS_DEREF_PROBLEM	0x24	Can not de-reference the alias.
LDAP_INAPPROPRIATE_AUTH	0x30	Authentication is inappropriate.
LDAP_INVALID_CREDENTIALS	0x31	The supplied credential is invalid.
LDAP_INSUFFICIENT_RIGHTS	0x32	The user has insufficient access right.
LDAP_BUSY	0x33	The server is busy.
LDAP_UNAVAILABLE	0x34	The server is unavailable.
LDAP_UNWILLING_TO_PERFORM	0x35	The server is not willing to handle directory requests.

Code	Value	Description
LDAP_LOOP_DETECT	0x36	The chain of referrals has looped back to a referring server.
LDAP_NAMING_VIOLATION	0x40	There was a naming violation.
LDAP_OBJECT_CLASS_VIOLATION	0x41	There was an object class violation.
LDAP_NOT_ALLOWED_ON_NONLEAF	0x42	Operation is not allowed on a non-leaf object.
LDAP_NOT_ALLOWED_ON_RDN	0x43	Operation is not allowed on RDN.
LDAP_ALREADY_EXISTS	0x44	The object already exists.
LDAP_NO_OBJECT_CLASS_MODS	0x45	Cannot modify object class.
LDAP_RESULTS_TOO_LARGE	0x46	Results returned are too large.
LDAP_AFFECTS_MULTIPLE_DSAS	0x47	Multiple directory service agents are affected.
LDAP_OTHER	0x50	Unknown error occurred.
LDAP_SERVER_DOWN	0x51	Cannot contact the LDAP server.
LDAP_LOCAL_ERROR	0x52	Local error occurred.
LDAP_ENCODING_ERROR	0x53	Encoding error occurred.
LDAP_DECODING_ERROR	0x54	Decoding error occurred.
LDAP_TIMEOUT	0x55	The search was timed out.
LDAP_AUTH_UNKNOWN	0x56	Unknown authentication error occurred.
LDAP_FILTER_ERROR	0x57	The search filter is incorrect.
LDAP_USER_CANCELLED	0x58	The user has canceled the operation.
LDAP_PARAM_ERROR	0x59	An incorrect parameter was passed to a routine.
LDAP_NO_MEMORY	0x5a	The system is out of memory.
LDAP_CONNECT_ERROR	0x5b	Cannot establish a connection to the server.
LDAP_NOT_SUPPORTED	0x5c	The feature is not supported.
LDAP_NO_RESULTS_RETURNED	0x5e	The feature is not supported.
LDAP_CONTROL_NOT_FOUND	0x5d	The ldap function (either ldap_parse_page_control or ldap_parse_sort_control) did not find the specified control.
LDAP_MORE_RESULTS_TO_RETURN	0x5f	Additional results are to be returned.
LDAP_CLIENT_LOOP	0x60	Client loop was detected.
LDAP_REFERRAL_LIMIT_EXCEEDED	0x61	The referral limit was exceeded.

Using a Network Trace to Interpret LDAP Errors

Through the use of a network trace, one can develop a better understanding of why an LDAP request did not perform as expected. By looking at the response packet to the query in question, one may gather additional data. For example, the following code shows the response from an LDAP Bind operation when incorrect credentials were passed to the DSA.

```
LDAP: Protocolop: BindResponse (1)
LDAP: MessageID
LDAP: Protocolop = BindResponse
LDAP: Result Code = Invalid Credentials LDAP: Matched DN =
LDAP: Error Message = 8009030C: LdapErr: DSID-0C090341, comment:
AcceptSecurityContext error, data 52e, v7bl.
```

The Result Code returned an LDAP error that translated to Invalid Credentials. The LDAP error codes may be found in a file called Winldap.h, which is a part of the Platform SDK. Additional information can be found by further drilling into the packet's data section when the DSA is a Microsoft Active Directory server. The error message returns the Win32 error code from the DSA. In this case 8009030C is returned which resolves to SEC_E_LOGON_DENIED. The DSID number is an ID that helps Microsoft Product Support Services debug the line of code from which the error was returned, if needed.

SUMMARY

LDAP is an open protocol standard that provides methods to manipulate data stored in network directories. Any combination of ADSI, the LDAP APIs, or a user searching the Active Directory through the User Interface (UI) results in the LDAP protocol being placed on the wire as the client communicates with the DSA. Because LDAP is a protocol standard it manifests itself in the same manner on the network wire, regardless of how the query or modification was generated at the application layer.

Appendix A: Examples

This section presents examples of uses of LDAP.

Creating External Cross-References

When you want to extend a search to a directory outside the forest, you can create a cross-reference object that references the external directory. To do this, you can use ADSI Edit or LDP.exe to create **crossRef** objects in the **Partitions** container that reference external directories.

You must provide values for the following attributes when you create a cross-reference object:

- **cn**: The name that describes the directory. For example, for the domain noam.reskit.com, your cn value could be "noam," or something else that describes that domain, such as "NorthAmerica."
- **nCName**: The distinguished name of the domain directory partition to which your cross-reference refers. If the domain name is noam.reskit.com, the value of nCName would be **dc=noam,dc=reskit,dc=com**.
- **dnsRoot**: The DNS host name of an LDAP server in the domain that is identified by nCName, for example, server1.noam.reskit.com. The value of dnsRoot also can be the domain name if you don't want to specify a server.

You must be able to resolve (ping) the name in dnsRoot. The dnsRoot is not necessarily another Windows 2000 system so it might not be a domain controller, but it may be the DNS address of an LDAP server instead. If the directory partition is a Windows 2000 domain from another forest, automatically generated knowledge should suffice; an external cross reference is not required.

To make use of cross-references, clients must be enabled to follow ("chase") referrals that are returned. Windows Address Book chases referrals by default. In the Ldp.exe application, you can specify **Chase Referrals** in the **Search** options. Programmatically using ADSI (for example, using ADO to search), you must specify whether to chase referrals.

To create a cross-reference to an external directory through a foreign container, you give the **nCName** attribute a value that is the name of the actual foreign directory. For example, an external LDAP directory might use X.500 naming such as **o=Organization Name,c=Country**, which would be used for the value of the **nCName** attribute. Queries for this directory must specify the foreign container object by name in the search base distinguished name. A request for a referral to such a location might come in the form of an LDAP URL embedded in a mail message or from an application that specifically names the directory distinguished name.

If you want a subtree search of a portion of your directory to always include a foreign LDAP directory that is not a Windows 2000 directory service, you can create a cross reference to the directory through a virtual container. To create a virtual container, give the **nCName** attribute of the cross reference object a value that is an immediate child object of an existing directory object and that also matches the DN of the foreign directory. Choose the location according to where you want the foreign directory to be locatable in Active Directory. A foreign directory that is stored on a Windows 2000 domain controller does not require an explicit cross-reference object. Because the domain component (dc=) portions of the distinguished names of all Windows 2000 domains match their DNS addresses, and because DNS is the worldwide namespace, all Windows 2000 domain controllers can generate external referrals to each other automatically.

Virtual containers are especially useful for smoothly integrating dynamic directories. For example, you might use an instant-messaging product such as Microsoft NetMeeting to publish a list of current or planned conference calls. LDAP is an effective protocol for querying such a published list; however, short-lived, highly volatile data is inappropriate for Active Directory storage. Therefore, you might use an in-memory, non-replicated LDAP server (one that can store volatile data) at an arbitrary point in the namespace. This volatile directory service then could be configured to inhabit a name inside your company's Active Directory namespace and could be made available to company users through a virtual container in Active Directory. This arrangement would allow users to find the list of conversations by directory tree navigation.

Suppose your domain name is reskit.com and you have installed your messaging application on a non-Active Directory LDAP server named Vds.it.reskit.com. On that server, you would create a directory for your volatile data, such as **cn=conversations,dc=reskit,dc=com**. Then on your Active Directory domain controller, you would create a cross-reference object and use the following attribute values:

- **cn=conversation server**
- **nCName=cn=conversations,dc=reskit,dc=com**
- **dnsRoot=vds.it.reskit.com**

When a user searches a subtree of **dc=reskit,dc=com**, the client receives results from the Windows 2000 domain controllers and a subordinate referral to the volatile directory service server at vds.it.reskit.com, which instructs the client to continue the LDAP search from **cn=conversations,dc=reskit,dc=com** and below on that server.

Demo.vbs

This section shows the Demo.vbs script code. This script creates an OU called demo that in turn contains three OUs, each of which includes several users.

```
Set rootDSE = GetObject("LDAP://RootDSE")
Set dom = GetObject("LDAP://" & rootDSE.Get("defaultNamingContext"))
DomainDn = dom.Get("distinguishedName")
wscript.echo DomainDn
```

' Create ou=demo

```
Set ouDemo = dom.Create("organizationalUnit", "OU=demo")
ouDemo.Description = "OU=Demo for Understanding LDAP"
ouDemo.setinfo
ouDemoDN = ouDemo.Get("distinguishedName")
wscript.echo ouDemoDN
```

' Create and populate ou=Research

```
Set ouResearch = ouDemo.Create("organizationalUnit", "OU=Research")
ouResearch.Description = "OU=Research Demo for Understanding LDAP"
ouResearch.setinfo
ouResearchDN = ouResearch.Get("distinguishedName")
wscript.echo ouResearchDN
```

' Create a user named Bob

```
Set usr = ouResearch.Create("user", "CN=Bob")
usr.Put "samAccountName", "Bob"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo
```

' Create a user named John

```
Set usr = ouResearch.Create("user", "CN=John")
usr.Put "samAccountName", "John"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo
```

' Create a user named Pam

```
Set usr = ouResearch.Create("user", "CN=Pam")
usr.Put "samAccountName", "Pam"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo
```

```
' Create a user named Steve
Set usr = ouResearch.Create("user", "CN=Steve")
usr.Put "samAccountName", "Steve"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo
```

```
' Create a user named Barney
Set usr = ouResearch.Create("user", "CN=Barney")
usr.Put "samAccountName", "Barney"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo
```

```
' Create a user named Tim
Set usr = ouResearch.Create("user", "CN=Tim")
usr.Put "samAccountName", "Tim"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create and populate ou=Sales
Set ousales = oudemo.Create("organizationalUnit", "OU=Sales")
ousales.Description = "OU=Sales Demo for Understanding LDAP"
ousales.setinfo
ousalesDN = ousales.Get("distinguishedName")
wscript.echo ousalesDN

' Create a user named Dan
Set usr = ousales.Create("user", "CN=Dan")
usr.Put "samAccountName", "Dan"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create a user named Micheal
Set usr = ousales.Create("user", "CN=Micheal")
usr.Put "samAccountName", "Micheal"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create a user named Lee
Set usr = ousales.Create("user", "CN=Lee")
usr.Put "samAccountName", "Lee"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create a user named Gary
Set usr = ousales.Create("user", "CN=Gary")
usr.Put "samAccountName", "Gary"
```

```
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create a user named Alice
Set usr = ousales.Create("user", "CN=Alice")
usr.Put "samAccountName", "Alice"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create a user named Jessica
Set usr = ousales.Create("user", "CN=Jessica")
usr.Put "samAccountName", "Jessica"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create and populate OU=Support
Set ouSupport = ouDemo.Create("organizationalUnit", "OU=Support")
ouSupport.Description = "OU=Support Demo for Understanding LDAP"
ouSupport.setinfo
ouSupportDN = ouSupport.Get("distinguishedName")
wscript.echo ouSupportDN

' Create a user named Brent
Set usr = ouSupport.Create("user", "CN=Brent")
usr.Put "samAccountName", "Brent"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
usr.SetInfo

' Create a user named Jim
Set usr = ouSupport.Create("user", "CN=Jim")
usr.Put "samAccountName", "Jim"
usr.setinfo
wscript.echo usr.Get("distinguishedName")
usr.AccountDisabled = False
```

```
usr.SetInfo
```

```
' Create a user named Curtis  
Set usr = ouSupport.Create("user", "CN=Curtis")  
usr.Put "samAccountName", "Curtis"  
usr.setinfo  
wscript.echo usr.Get("distinguishedName")  
usr.AccountDisabled = False  
usr.SetInfo
```

```
' Create a user named Mark  
Set usr = ouSupport.Create("user", "CN=Mark")  
usr.Put "samAccountName", "Mark"  
usr.setinfo  
wscript.echo usr.Get("distinguishedName")  
usr.AccountDisabled = False  
usr.SetInfo
```

Appendix B: RFC Reference

This section lists RFCs pertaining to the Lightweight Directory Access Protocol versions 2 and 3.

RFC	Title
RFC 1510	"The Kerberos Network Authentication Service (v5)"
RFC 1777	"Lightweight Directory Access Protocol"
RFC 1778	"The String Representation of Standard Attribute Syntaxes"
RFC 1779	"A String Representation of Distinguished Names"
RFC 1823	"The LDAP Application Program Interface"
RFC 1960	"A String Representation of LDAP Search Filters"
RFC 2251	"Lightweight Directory Access Protocol (v3)"
RFC 2251	"Lightweight Directory Access Protocol (v3)"
RFC 2252	"Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions"
RFC 2254	"The String Representation of LDAP Search Filters"
RFC 2255	"The LDAP URL Format"
RFC 2256	"A Summary of the X.500(96) User Schema for Use with LDAPv3"

For More Information

For the latest information on Windows 2000 Server, check out our Web site at <http://www.microsoft.com/windows2000> and the Windows 2000/NT Forum at <http://computingcentral.msn.com/topics/windowsnt>.