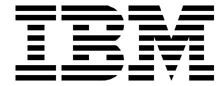
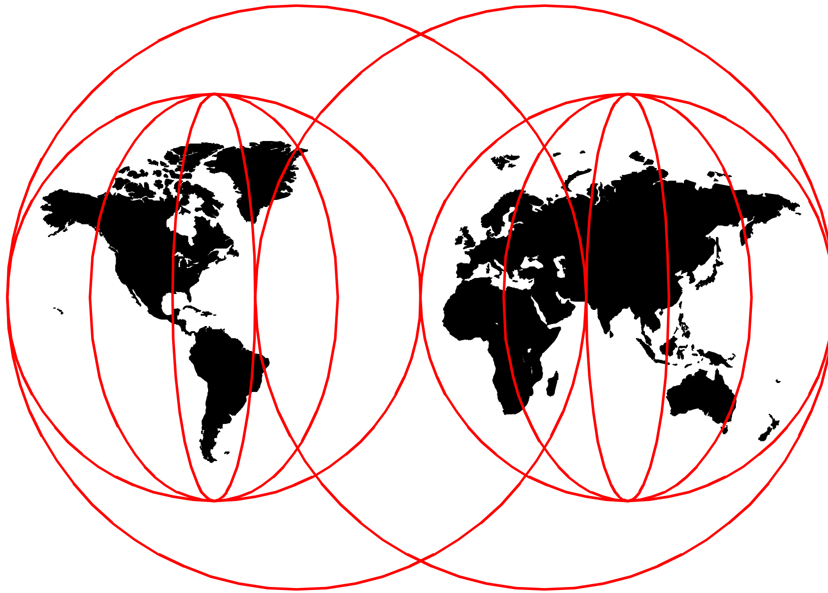


Understanding LDAP

Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-4986-00

Contents

Figures vii

Tables ix

Preface xi
The Team That Wrote This Redbook xi
Comments Welcome xii

Chapter 1. LDAP: The New Common Directory 1
1.1 What is a Directory? 2
1.1.1 Differences Between Directories and Databases 2
1.1.2 Directory Clients and Servers 4
1.1.3 Distributed Directories 6
1.1.4 Directory Security 7
1.2 The Directory as Infrastructure 8
1.2.1 Directory-Enabled Applications 8
1.2.2 The Benefits of a Common Directory 9
1.3 LDAP History and Standards 10
1.3.1 OSI and the Internet 10
1.3.2 X.500: The Directory Service Standard 11
1.3.3 LDAP: Lightweight Access to X.500 12
1.4 LDAP: Protocol or Directory? 14
1.5 The LDAP Road Map 15
1.6 The Quick Start: A Public LDAP Example 16

Chapter 2. LDAP Concepts and Architecture 19
2.1 Overview of LDAP Architecture 19
2.2 The LDAP Models 24
2.2.1 The Information Model 25
2.2.2 The Naming Model 28
2.2.3 The Functional Model 35
2.2.4 The Security Model 42
2.3 Security 43
2.3.1 No Authentication 44
2.3.2 Basic Authentication 44
2.3.3 Simple Authentication and Security Layer (SASL) 45
2.4 Manageability 49
2.4.1 LDAP Command Line Tools 50
2.4.2 LDAP Data Interchange Format (LDIF) 50
2.5 Platform Support 56

© Copyright IBM Corp. 1998

Chapter 3. Designing and Maintaining an LDAP Directory 57
3.1 Directory Design Guidelines 57
3.1.1 Defining the Data Model 58
3.1.2 Security Policy 65
3.1.3 Physical Design 69
3.2 Migration Planning 73
3.3 Example Scenarios 76
3.3.1 Small Organization 76
3.3.2 Large Organization 79

Chapter 4. Building LDAP-Enabled Applications 85
4.1 LDAP Software Development Kits (SDKs) 86
4.2 The C Language API to LDAP 86
4.2.1 Getting Started 86
4.2.2 Synchronous and Asynchronous Use of the API 91
4.2.3 A Synchronous Search Example 92
4.2.4 More about Search Filters 96
4.2.5 Parsing Search Results 96
4.2.6 An Asynchronous Example 99
4.2.7 Error Handling 104
4.2.8 Authentication Methods 108
4.2.9 Multithreaded Applications 113
4.3 LDAP Command Line Tools 115
4.3.1 The Search Tool: ldapsearch 116
4.3.2 The ldapmodify and ldapadd Utilities 117
4.3.3 The ldapdelete Tool 118
4.3.4 The ldapmodrdn Tool 119
4.3.5 Security Considerations 119
4.4 LDAP URLs 120
4.4.1 Uses of LDAP URLs 122
4.4.2 LDAP URL APIs 123
4.5 The Java Naming and Directory Interface (JNDI) 124
4.5.1 JNDI Example Program 127

Chapter 5. The Future of LDAP 131
5.1 The IETF LDAP Road Map 131
5.1.1 Access Control Requirements for LDAP 132
5.1.2 Scrolling View Browsing of Search Results 133
5.1.3 LDAP Clients Finding LDAP Servers 133
5.2 Distributed Computing Environment (DCE) and LDAP 133
5.2.1 LDAP Interface for the GDA 135
5.2.2 LDAP Interface for the CDS 135
5.2.3 Future LDAP Integration 136

5.3 Other Middleware Software 137
5.4 The Directory-Enabled Networks Initiative 138

Appendix A. Other LDAP References 139
A.1 The Internet Engineering Task Force (IETF) 139
A.2 The University of Michigan (UMICH) 140
A.3 Software Development Kits. 140
A.4 Other Sources. 140
A.4.1 Vendors Mentioned in this Book 141
A.4.2 LDAP, General 141
A.4.3 Request for Comments (RFCs) 142
A.4.4 Security. 142

Appendix B. LDAP Products and Services 143
B.1 IBM Product Offerings. 143
B.1.1 IBM eNetwork LDAP Directory 143
B.1.2 IBM eNetwork X.500 Directory for AIX 144
B.1.3 IBM eNetwork LDAP Client Pack for Multiplatforms 145
B.2 Lotus Domino 146
B.3 Tivoli User Administration: LDAP Endpoint. 147
B.4 Other LDAP Server Products 148
B.4.1 Netscape Directory Server 148
B.4.2 Novell LDAP Services for NDS 149
B.4.3 Microsoft Active Directory 149
B.5 LDAP Enabled Clients and Applications 150
B.6 LDAP Development Kits and Tools. 150
B.7 Public LDAP Services 151

Appendix C. LDAP C Language API Functions and Error Codes 153
C.1 C Language API Calls 153
C.1.1 Functions to Establish and Terminate a Connection 153
C.1.2 Session-Handling Functions. 154
C.1.3 Interacting with the Server 154
C.1.4 Error Handling 155
C.1.5 Analyzing Results. 156
C.1.6 Freeing Memory 157
C.1.7 Other Functions 157
C.2 LDAP API Error Codes 158

Appendix D. Special Notices 161

Appendix E. Related Publications 163
E.1 International Technical Support Organization Publications 163
E.2 Redbooks on CD-ROMs 163

E.3 Other Publications 164

How to Get ITSO Redbooks 165
How IBM Employees Can Get ITSO Redbooks 165
How Customers Can Get ITSO Redbooks 166
IBM Redbook Order Form 167

List of Abbreviations 169

Index 171

ITSO Redbook Evaluation 177

Chapter 2. LDAP Concepts and Architecture

LDAP is based on the client/server model of distributed computing (see 1.1.2, "Directory Clients and Servers" on page 4). LDAP has evolved as a lightweight protocol for accessing information in X.500 directory services. It has since become more independent of X.500, and servers that specifically support the LDAP protocol rather than the X.500 Directory Access Protocol (DAP) are now common. The success of LDAP has been largely due to the following characteristics that make it simpler to implement and use, compared to X.500 and DAP:

- LDAP runs over TCP/IP rather than the OSI protocol stack. TCP/IP is less resource-intensive and is much more widely available, especially on desktop systems.
- The functional model of LDAP is simpler. It omits duplicate, rarely-used and esoteric features. This makes LDAP easier to understand and to implement.
- LDAP uses strings to represent data rather than complicated structured syntaxes such as ASN.1 (Abstract Syntax Notation One).

This chapter explains the basic architecture of LDAP. It discusses the information, naming, functional, and security models that form the basis of the LDAP architecture. Various terms and concepts defined by or needed to understand the LDAP architecture are introduced along the way. After a general overview of the architecture, each of the models that form the backbone of the LDAP architecture is discussed in detail.

2.1 Overview of LDAP Architecture

LDAP defines the content of messages exchanged between an LDAP client and an LDAP server. The messages specify the operations requested by the client (search, modify, delete, and so on), the responses from the server, and the format of data carried in the messages. LDAP messages are carried over TCP/IP, a connection-oriented protocol; so there are also operations to establish and disconnect a session between the client and server.

However, for the designer of an LDAP directory, it is not so much the structure of the messages being sent and received over the wire that is of interest. What is important is the logical model that is defined by these messages and data types, how the directory is organized, what operations are possible, how information is protected, and so forth.

© Copyright IBM Corp. 1998

The general interaction between an LDAP client and an LDAP server takes the following form:

- The client establishes a session with an LDAP server. This is known as *binding* to the server. The client specifies the host name or IP address and TCP/IP port number where the LDAP server is listening. The client can provide a user name and a password to properly authenticate with the server. Or the client can establish an anonymous session with default access rights. The client and server can also establish a session that uses stronger security methods such as encryption of data.
- The client then performs operations on directory data. LDAP offers both read and update capabilities. This allows directory information to be managed as well as queried. LDAP also supports searching the directory for data meeting arbitrary user-specified criteria. Searching is a very common operation in LDAP. A user can specify what part of the directory to search and what information to return. A search filter that uses Boolean conditions specifies what directory data matches the search.
- When the client is finished making requests, it closes the session with the server. This is also known as *unbinding*.

Although it is not defined by the LDAP protocol and architecture itself, there is a well-known LDAP API (application program interface) that allows applications to easily interact with LDAP servers. The API can be considered an extension to the LDAP architecture. Although the C language LDAP API is only an informational RFC and the most recent update to it is an Internet Draft, it has achieved de facto standard status because it is supported by all major LDAP vendors. The philosophy of the LDAP API is to keep simple things simple. This means that adding directory support to existing applications can be done with low overhead. As we will see in Chapter 4, “Building LDAP-Enabled Applications” on page 85, this interface is reasonably easy to use and implement in applications.

Because LDAP was originally intended as a lightweight alternative to DAP for accessing X.500 directories, it follows an X.500 model (see 1.3.2, “X.500: The Directory Service Standard” on page 11). The directory stores and organizes data structures known as *entries*.

A directory entry usually describes an object such as a person, a printer, a server, and so on. Each entry has a name called a distinguished name (DN) that uniquely identifies it. The DN consists of a sequence of parts called relative distinguished names (RDNs), much like a file name consists of a path of directory names in many operating systems such as UNIX and Windows. The entries can be arranged into a hierarchical tree-like structure based on

their distinguished names. This tree of directory entries is called the Directory Information Tree (DIT).

Each entry contains one or more attributes that describe the entry. Each attribute has a type and a value. For example, the directory entry for a person might have an attribute called `telephonNnumber`. The syntax of the `telephoneNumber` attribute would specify that a telephone number must be a string of numbers that can contain spaces and hyphens. The value of the attribute would be the person's telephone number, such as 512-555-1212.

A directory entry describes some object. An object class is a general description, sometimes called a template, of an object as opposed to the description of a particular object. For instance, the object class person has a `surname` attribute, whereas the object describing John Smith has a surname attribute with the value Smith. The object classes that a directory server can store and the attributes they contain are described by schema. Schema define what object classes are allowed where in the directory, what attributes they must contain, what attributes are optional, and the syntax of each attribute. For example, a schema could define a person object class. The person schema might require that a person have a `surname` attribute that is a character string, specify that a person entry can optionally have a `telephoneNumber` attribute that is a string of numbers with spaces and hyphens, and so on.

LDAP defines operations for accessing and modifying directory entries such as:

- Searching for entries meeting user-specified criteria
- Adding an entry
- Deleting an entry
- Modifying an entry
- Modifying the distinguished name or relative distinguished name of an entry (move)
- Comparing an entry

LDAP is documented in several IETF RFCs. As discussed in 1.3.3, "LDAP: Lightweight Access to X.500" on page 12, the current version of LDAP is Version 3. That section also lists the RFCs associated with each version of LDAP.

The LDAP Version 3 RFCs are again listed below along with a short description to provide an overview of the documents defining the LDAP architecture.

1. RFC 2251 *Lightweight Directory Access Protocol (v3)*

   Describes the LDAP protocol designed to provide lightweight access to directories supporting the X.500 model. The lightweight protocol is meant to be implementable in resource-constrained environments such as browsers and small desktop systems. This RFC is the core of the LDAP family of RFCs. It describes how entries are named with distinguished names, defines the format of messages exchanged between client and server, enumerates the operations that can be performed by the client, and specifies that data is represented using UTF-8 character encoding.

   The RFC specifies that the schema describing directory entries must themselves be readable so that a client can determine what type of objects a directory server stores. It defines how the client can be referred to another LDAP server if a server does not contain the requested information. It describes how individual operations can be extended using controls and how additional operations can be defined using extensions. It also discusses how clients can authenticate to servers and optionally use Simple Authentication and Security Layer (SASL) to allow additional authentication mechanisms.

2. RFC 2252 *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*

   LDAP uses octet strings to represent the values of attributes for transmission in the LDAP protocol. This RFC defines how values such as integers, time stamps, mail addresses, and so on are represented. For example, the integer 123 is represented by the string "123". These definitions are called attribute syntaxes. This RFC describes how an attribute with a syntax such as “telephone number” is encoded. It also defines matching rules to determine if values meet search criteria. An example is `caseIgnoreString`, which is used to compare character strings when case is not important.

   These attribute types and syntaxes are used to build schema that describe objects classes. A schema lists what attributes a directory entry must or may have. Every directory entry has an `objectclass` attribute that lists the (one or more) schema that describe the entry. For example, a directory entry could be described by the object classes `residentialPerson` and `organizationalPerson`. If an `objectclass` attribute includes the value `extensibleObject`, it can contain any attribute.

3. RFC 2253 *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*

   Distinguished names (DNs) are the unique identifiers, sometimes called primary keys, of directory entries. X.500 uses ASN.1 to encode

distinguished names. LDAP encodes distinguished names as strings. This RFC defines how distinguished names are represented as strings. A string representation is easy to encode and decode and is also human readable. A DN is composed of a sequence of relative distinguished names (RDNs) separated by commas. The sequence of RDNs making up a DN names the ancestors of a directory entry up to the root of the DIT. Each RDN is composed of an attribute value from the directory entry. For example, the DN `cn=John Smith,ou=Austin,o=IBM,c=US` represents a directory entry for a person with the `common name` (`cn`) John Smith under the `organizational unit` (`ou`) Austin in the `organization` (`o`) IBM in the `country` (`c`) US.

4. RFC 2254 *The String Representation of LDAP Search Filters*

   LDAP search filters provide a powerful mechanism to search a directory for entries that match specific criteria. The LDAP protocol defines the network representation of a search filter. This document defines how to represent a search filter as a human-readable string. Such a representation can be used by applications or in program source code to specify search criteria. Attribute values are compared using relational operators such as equal, greater than, or “sounds like” for approximate or phonetic matching. Boolean operators can be used to build more complex search filters. For example, the search filter (| (sn=Smith) (cn=Jo*)) searches for entries that either have a `surname` attribute of Smith or that have a `common name` attribute that begins with Jo.

5. RFC 2255 *The LDAP URL Format*

   Uniform Resource Locators (URLs) are used to identify Web pages, files, and other resources on the Internet. An LDAP URL specifies an LDAP search to be performed at a particular LDAP server. An LDAP URL represents in a compact and standard way the information returned as the result of the search. Section 4.4, “LDAP URLs” on page 120, explains LDAP URLs in detail.

6. RFC 2256 *A Summary of the X.500(96) User Schema for use with LDAPv3*

   Many schema and attributes commonly accessed by directory clients are already defined by X.500. This RFC provides an overview of those attribute types and object classes that LDAP servers should recognize. For instance, attributes such as `cn` (`common name`), `description`, and `postalAddress` are defined. Object classes such as country, organizationalUnit, groupOfNames, and applicationEntity are also defined.

The RFCs listed above build up the core LDAP Version 3 specification. In addition to these RFCs, the IETF lists a number of so-called proposed extensions to LDAP Version 3 that vendors may implement as well. However, these proposed extensions only have the status of Internet Drafts and may

therefore still change. The following list summarizes some of these proposed extensions:

- Mandatory-to-Implement Authentication

  An attempt to have at least one standard, secure authentication method available in all servers and clients (not only LDAP), rather than individual methods for each protocol above TCP/IP.

- Extensions for Dynamic Directory Services

  This is a protocol extension that allows clients to interact more reliably with servers while directory contents are being changed.

- Use of Language Codes in LDAP

  Describes the addition of natural language codes to attributes stored in an LDAP directory.

- LDAPv3 Extension for Transport Layer Security

  Defines the integration of the Transport Layer Security (TLS) mechanism into LDAP.

- LDAP Control Extension for Simple Paged Results Manipulation

  Describes a control extension for paging of search results. This is of special value for simple, limited-function clients so they can request that search results are returned in smaller portions (pages) at a time.

- Referrals and Knowledge References in LDAP Directories

  Defines how referrals and reference information can be stored as attributes and how they may be used.

- LDAP Control Extension for Server Side Sorting of Search Results

  Allows sorting of search results on the server rather than on the client. This may be desirable to build simpler, limited function clients.

- The LDAP Application Program Interface

  Defines the C language application program interface (API) to LDAP. Most vendors already incorporate this extension, or at least a subset of it. See Chapter 4, “Building LDAP-Enabled Applications” on page 85, for more information on the C language API.

2.2 The LDAP Models

LDAP can be better understood by considering the four models upon which it is based:

Information Describes the structure of information stored in an LDAP directory.

Naming Describes how information in an LDAP directory is organized and identified.

Functional Describes what operations can be performed on the information stored in an LDAP directory.

Security Describes how the information in an LDAP directory can be protected from unauthorized access.

The following sections discuss the four LDAP models.

2.2.1 The Information Model

The basic unit of information stored in the directory is called an entry. Entries represent objects of interest in the real world such as people, servers, organizations, and so on. Entries are composed of a collection of attributes that contain information about the object. Every attribute has a type and one or more values. The type of the attribute is associated with a syntax. The syntax specifies what kind of values can be stored. For example, an entry might have a `facsimilieTelephoneNumber` attribute. The syntax associated with this type of attribute would specify that the values are telephone numbers represented as printable strings optionally followed by keywords describing paper size and resolution characteristics. It is possible that the directory entry for an organization would contain multiple values in this attribute—that is that an organization or person represented by the entity would have multiple fax numbers. The relationship between a directory entry and its attributes and their values is shown in Figure 6.

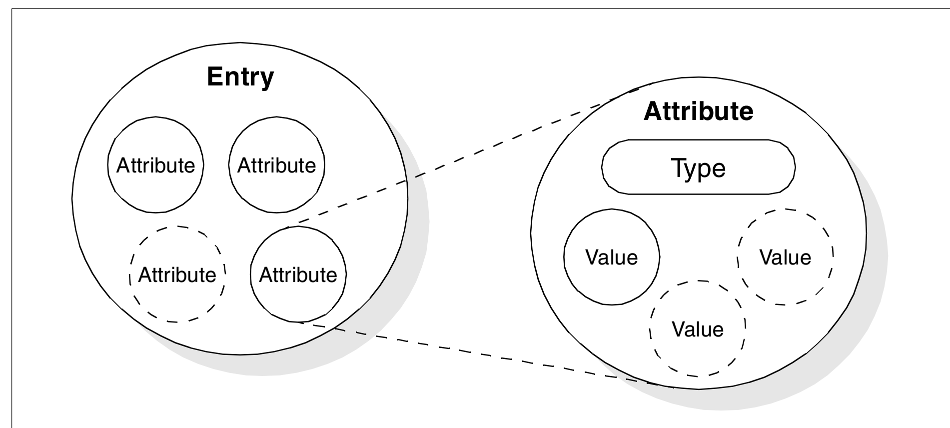


Figure 6. Entries, Attributes and Values

In addition to defining what data can be stored as the value of an attribute, an attribute syntax also defines how those values behave during searches and other directory operations. The attribute `telephoneNumber`, for example, has a syntax that specifies:

- Lexicographic ordering.
- Case, spaces and dashes are ignored during the comparisons.
- Values must be character strings.

For example, using the correct definitions, the telephone numbers “512-838-6008”, “512838-6008”, and “5128386008” are considered the same. A few of the syntaxes that have been defined for LDAP are listed in the following table.

Table 2. Some of the LDAP Attribute Syntaxes

Syntax	Description
bin	Binary information.
ces	Case exact string, also known as a "directory string", case is significant during comparisons.
cis	Case ignore string. Case is not significant durring comparisons.
tel	Telephone number. The numbers are treated as text, but all blanks and dashes are ignored.
dn	Distinguished name.
Generalized Time	Year, month, day, and time represented as a printable string.
Postal Address	Postal address with lines separated by "$" characters.

Table 3 lists some common attributes. Some attributes have alias names that can be used wherever the full attribute name is used. For example, `cn` can be used when referring to the attribute `commonName`.

Table 3. Common LDAP Attributes

Attribute, Alias	Syntax	Description	Example
commonName, cn	cis	Common name of an entry	John Smith
surname, sn	cis	Surname (last name) of a person	Smith
telephoneNumber	tel	Telephone number	512-838-6008

Attribute, Alias	Syntax	Description	Example
organizationalUnitName, ou	cis	Name of an organizational unit	itso
owner	dn	Distinguished name of the person that owns the entry	cn=John Smith, o=IBM, c=US
organization, o	cis	Name of an organization	IBM
jpegPhoto	bin	Photographic image in JPEG format	Photograph of John Smith

Constraints can be associated with attribute types to limit the number of values that can be stored in the attribute or to limit the total size of a value. For example, an attribute that contains a photo could be limited to a size of 10 KB to prevent the use of unreasonable amounts of storage space. Or an attribute used to store a social security number could be limited to holding a single value.

Schemas define the type of objects that can be stored in the directory. Schemas also list the attributes of each object type and whether these attributes are required or optional. For example, in the person schema, the attribute `surname (sn)` is required, but the attribute `description` is optional. Schema-checking ensures that all required attributes for an entry are present before an entry is stored. Schema-checking also ensures that attributes not in the schema are not stored in the entry. Optional attributes can be filled in at any time. Schema also define the inheritance and subclassing of objects and where in the DIT structure (hierarchy) objects may appear.

Table 4 lists a few of the common schema (object classes and their required attributes). In many cases, an entry can consist of more than one object class:

Table 4. Object Classes and Required Attributes

Object Class	Description	Required Attributes
InetOrgPerson	Defines entries for a person	commonName (cn) surname (sn) objectClass
organizationalUnit	Defines entries for organizational units	ou objectClass
organization	Defines entries for organizations	o objectClass

Though each server can define its own schema, for interoperability it is expected that many common schema will be standardized (refer to RFC 2252, *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*, and RFC 2256, *A Summary of the X.500(96) User Schema for use with LDAPv3*).

There are times when new schema will be needed at a particular server or within an organization. In LDAP Version 3, a server is required to return information about itself, including the schema that it uses. A program can therefore query a server to determine the contents of the schema. This server information is stored at the special zero-length DN (see 2.2.2, “The Naming Model” on page 28, for more details).

Objects can be derived from other objects. This is known as subclassing. For example, suppose an object called person was defined that included a surname and so on. An object class organizationalPerson could be defined as a subclass of the person object class. The organizationPerson object class would have the same attributes as the person object class and could add other attributes such as `title` and `officenumber`. The person object class would be called the superior of the organizationPerson object class. One special object class, called top, has no superiors. The top object class includes the mandatory `objectClass` attribute. Attributes in top appear in all directory entries as specified (required or optional).

Each directory entry has a special attribute called objectClass. The value of the `objectClass` attribute is a list of two or more schema names. These schema define what type of object(s) the entry represents. One of the values must be either top or alias. Alias is used if the entry is an alias for another entry (see 2.2.2, “The Naming Model” on page 28), otherwise top is used. The `objectClass` attribute determines what attributes the entry must and may have.

The special object class `extensibleObject` allows any attribute to be stored in the entry. This can be more convenient than defining a new object class to add a special attribute to a few entries, but also opens up that object to be able to contain anything (which might not be a good thing in a structured system).

2.2.2 The Naming Model

The LDAP naming model defines how entries are identified and organized. Entries are organized in a tree-like structure called the Directory Information Tree (DIT). Entries are arranged within the DIT based on their distinguished name (DN). A DN is a unique name that unambiguously identifies a single

entry. DNs are made up of a sequence of relative distinguished names (RDNs). Each RDN in a DN corresponds to a branch in the DIT leading from the root of the DIT to the directory entry.

Each RDN is derived from the attributes of the directory entry. In the simple and common case, an RDN has the form <attribute name> = <value> (see Figure 8 on page 31 for the complete syntax of DNs and RDNs). A DN is composed of a sequence of RDNs separated by commas.

An example of a DIT is shown in Figure 7. The example is very simple, but can be used to illustrate some basic concepts. Each box represents a directory entry. The root directory entry is conceptual, but does not actually exist. Attributes are listed inside each entry. The list of attributes shown is not complete. For example, the entry for the country DE (`c=DE`) could have an attribute called `description` with the value `Germany`.

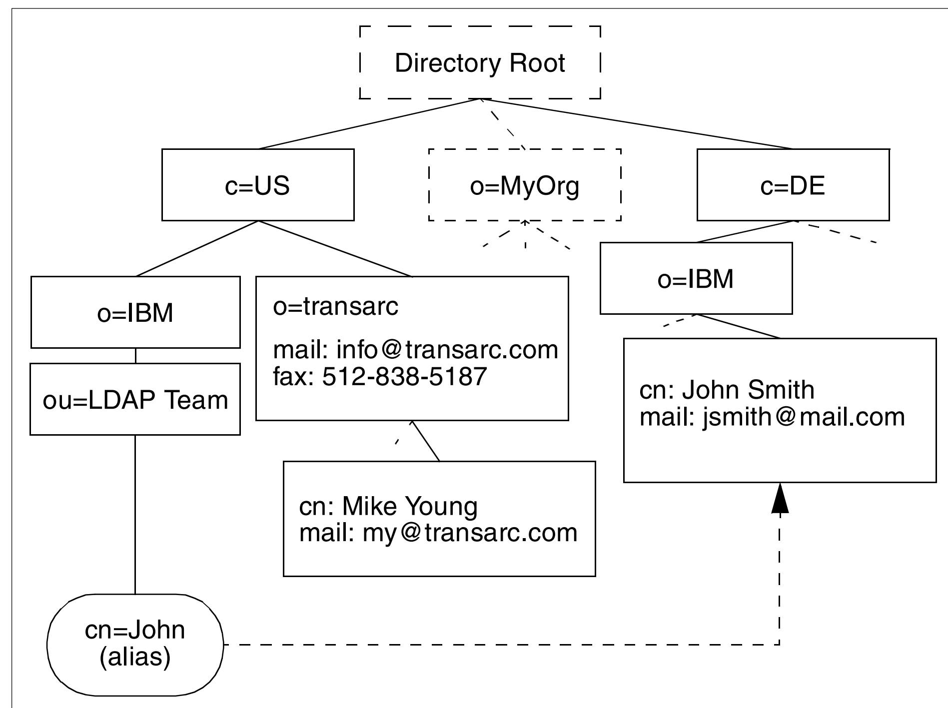


Figure 7. Example Directory Information Tree (DIT)

The organization of the entries in the DIT are restricted by their corresponding object class definitions. It is usual to follow either a geographical or an organizational scheme. For example, entries that

represent countries would be at the top of the DIT. Below the countries would be national organizations, states, and provinces, and so on. Below this level, entries might represent people within those organizations or further subdivisions of the organization. The lowest layers of the DIT entries could represent any object, such as people, printers, application servers, and so on. The depth or breadth of the DIT is not restricted and can be designed to suit application requirements. See Chapter 3, “Designing and Maintaining an LDAP Directory” on page 57, for information on designing a DIT.

Entries are named according to their position in the DIT. The directory entry in the lower-right corner of Figure 7 has the DN `cn=John Smith,o=IBM,c=DE`. Note that DNs read from leaf to root as opposed to file system names which usually read from root to leaf. The DN is made up of a sequence of RDNs. Each RDN is constructed from an attribute (or attributes) of the entry it names. For example, the DN `cn=John Smith,o=IBM,c=DE` is constructed by adding the RDN `cn=John Smith` to the DN of the ancestor entry `o=IBM,c=DE`. Note that `cn=John Smith` is an attribute in the entry `cn=John Smith,o=IBM,c=DE`. The DN of an entry is specified when it is created. It would have been legal, though not intuitive, to have created the entry with the DN `mail=jsmith@mail.com,o=IBM,c=DE`.

The DIT is described as being tree-like implying it is not a tree. This is because of aliases. Aliases allow the tree structure to be circumvented. This can be useful if an entry belongs to more than one organization or if a commonly used DN is too complex. Another common use of aliases is when entries are moved within the DIT and you want access to continue to work as before. In Figure 7, `cn=John,ou=LDAP Team,o=IBM,c=US` is an alias for `cn=John Smith,o=IBM,c=DE`. Aliases do not have to point to leaf entries in the DIT. For example, `o=Redbook,c=US` could be an alias for `ou=ITSO,o=IBM,c=US`.

2.2.2.1 Distinguished Name Syntax

DNs are used as primary keys to entries in the directory. LDAP defines a user-oriented string representation of DNs. The syntax of DNs, which consist of a sequence of RDNs, was described informally above. Figure 8 on page 31 shows the formal grammar of DNs.

Note that RDNs can be more complicated than in the examples shown above. An RDN can be composed of multiple attributes joined by “+” as in the DN `cn=John Smith+l=Stuttgart,o=IBM,c=DE`.

If attribute values contain special characters or leading or trailing spaces, those characters must be escaped by preceding them with a backslash character. The following DN contains a comma character `o=Transarc\, Inc.,c=US`.

DNs in LDAP Version 3 are more restrictive than in LDAP V2. For example, in LDAP V2, semicolons could also be used to separate RDNs. LDAP V3 must accept the older syntax, but must not generate DNs that do not conform to the newer syntax. The exact grammar for a distinguished name syntax is shown in Figure 8.

```
distinguishedName = [name]                    ; may be empty string

name       = name-component *("," name-component)

name-component = attributeTypeAndValue *("+" attributeTypeAndValue)

attributeTypeAndValue = attributeType "=" attributeValue

attributeType = (ALPHA 1*keychar) / oid
keychar    = ALPHA / DIGIT / "-"

oid        = 1*DIGIT *("." 1*DIGIT)

attributeValue = string

string     = *( stringchar / pair )
             / "#" hexstring
             / QUOTATION *( quotechar / pair ) QUOTATION ; only from v2

quotechar     = <any character except "\" or QUOTATION >

special    = "," / "=" / "+" / "<" /  ">" / "#" / ";"

pair       = "\" ( special / "\" / QUOTATION / hexpair )
stringchar = <any character except one of special, "\" or QUOTATION >

hexstring  = 1*hexpair
hexpair    = hexchar hexchar

hexchar    = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
             / "a" / "b" / "c" / "d" / "e" / "f"

ALPHA      =  <any ASCII alphabetic character>     ; (decimal 65-90 and 97-122)
DIGIT      =  <any ASCII decimal digit>            ; (decimal 48-57)
QUOTATION  =  <the ASCII double quotation mark character '"' decimal 34>
```

Figure 8. Distinguished Name Grammar

The attribute types used in the RDN can be represented by a dotted decimal string encoding of its object identifier. For example, `cn=John` could also be written as `2.5.4.2=John`. However, frequently used attribute names have a string representation that is obviously easier to understand. Table 5 lists some of the common attribute types and their string representation. Please

notice that because attribute names are case insensitive, you might see different uppercase/lowercase notations in the literature.

Table 5. Attribute Type String Representations

Attribute Type	String
CommonName	CN
LocalityName	L
StateOrProvinceName	ST
OrganizationName	O
OrganizationalUnitName	OU
CountryName	C
StreetAddress	STREET
domainComponent	DC
userid	UID

2.2.2.2 Suffixes and Referrals

An individual LDAP server might not store the entire DIT. A server might store the entries for a particular department and not the entries for the ancestors of the department. For example, a server might store the entries for the ITSO department at IBM. The highest node in the DIT stored by the server would be `ou=ITSO,o=IBM,c=US`. The server would not store entries for `c=US` or for `o=IBM,c=US`. The highest entry stored by a server is called a suffix. Each entry stored by the server ends with this suffix (remember that in the DN syntax, the higher-level entries are at the end).

A server can support multiple suffixes. For example, in addition to storing information about the ITSO department, the same server could store information about the sales department at Transarc. The server would then have the suffixes `ou=ITSO,o=IBM,c=US` and `ou=sales,o=Transarc,c=US`.

Since a server might not store the entire DIT, servers need to be linked together in some way in order to form a distributed directory that contains the entire DIT. This is accomplished with referrals. Continuing the example, another server might store the entry `o=IBM,c=US` but not information about the ITSO department. If somebody searched this directory server for information about the ITSO department, no information would be found. However, the server can store a referral to the LDAP server that does contain the information. This referral acts like a pointer that can be followed to where the desired information is stored. Such an example is shown in Figure 9, where

the referral arrow shows the logical connection of a referral and does not reflect the technical implementaion (see text that follows).

A referral is an entry of `objectClass` referral. It has an attribute, `ref`, whose value is the LDAP URL of the referred entry on another LDAP server. See 4.4, "LDAP URLs" on page 120, for information about LDAP URLs.

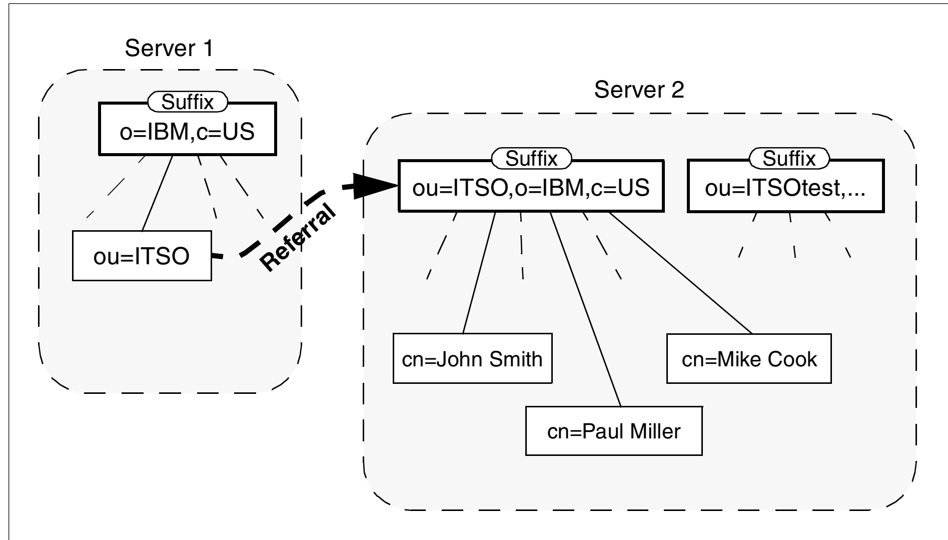


Figure 9. Example DIT Showing Suffixes and Referrals

When a client sends a request to an LDAP server, the response to the client may be a referral. The client can then choose to follow the referral by querying the other LDAP server contained in the referral returned by the first LDAP server. Referrals are not followed (resolved) by servers. This can improve server performance by off-loading the work of contacting other servers to the client.

Figure 10 illustrates a client following a referral. An LDAP client requests information from LDAP Server 1 (1). This request is answered with a referral to LDAP Server 2 (2). The LDAP client then contacts LDAP Server 2 (3). LDAP Server 2 provides the requested data to the client (4).

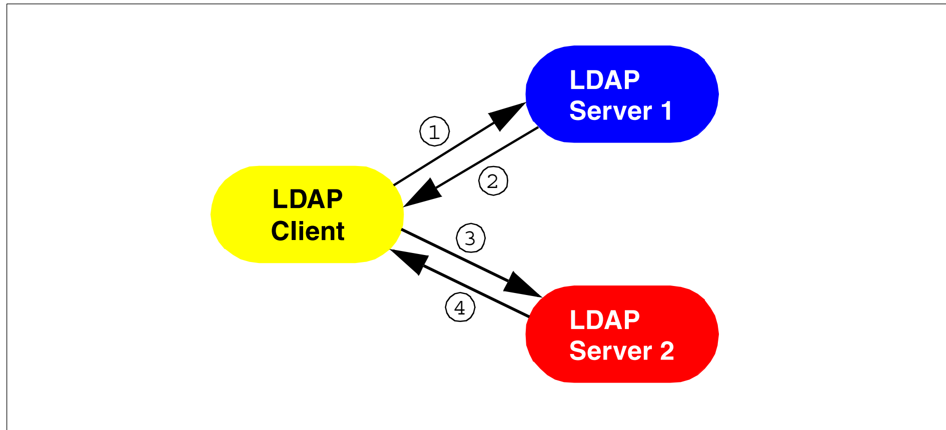


Figure 10. Referral Followed by Client

Figure 11 illustrates chaining. An LDAP client requests information from LDAP Server 1 (1). LDAP Server 1 finds a referral to Server 2 and forwards the request (2). Server 2 provides the requested data to LDAP Server 1 (3). LDAP Server 1 then returns the result to the client (4). Note that this explanation and Figure 11 are for illustration purposes only since chaining is not included in either the LDAP Version 2 or Version 3 specifications.

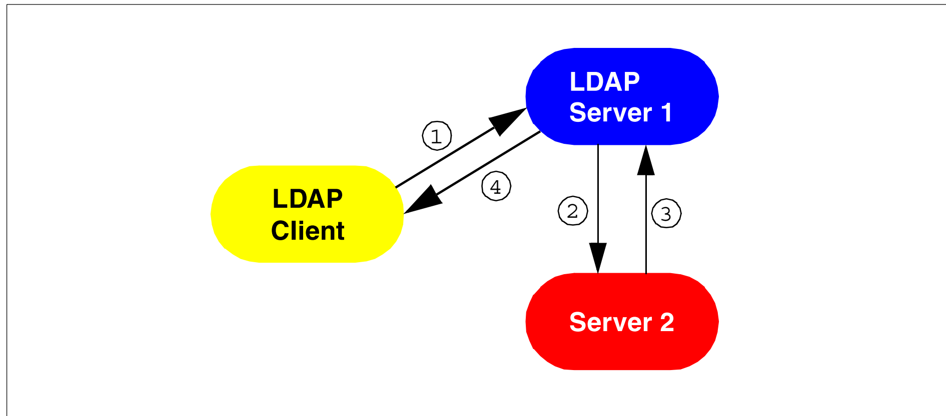


Figure 11. Server Chaining

The LDAP API allows the programmer to specify whether returned referrals should be followed automatically or returned to the program. If referrals are followed automatically, the LDAP client library (not the server nor the application program) follows the referral. This requires no extra coding and is transparent to the programmer. To prevent lengthy searches or referrals that

(mistakenly) form a loop, the programmer can limit the number of referrals followed for a request.

If the referral is returned to the program, code must be supplied to recognize that a referral has been returned. The referral can be examined and a decision made whether to follow it or not. This is more complicated, but gives the programmer greater choice of which referrals to follow.

Referrals allow a DIT to be partitioned and distributed across multiple servers. Portions of the DIT can also be replicated. This can improve performance and availability. See Chapter 3, “Designing and Maintaining an LDAP Directory” on page 57, for information on designing a distributed directory.

LDAP Version 2 did not formally define referrals, but Version 3 does include them. Neither Version 2 nor Version 3 define chaining, but it is not prohibited if vendors chose to implement it. Vendors, for example, may chose to implement an X.500-type chaining mechanism or functionality provided by distributed databases to achieve this.

2.2.2.3 Server Information

An LDAP Version 3 server must provide information about itself. The special entry called the root DSE with a zero-length (empty) DN contains attributes that describe the server. These attributes can be retrieved to discover basic information about the server and the DIT that it stores. Server-specific information available includes:

- The suffixes, also called naming contexts, the server stores
- The DN of a special entry that contains a list of all the objectClass and attribute schema known to the server
- The version(s) of LDAP supported,
- A list of supported extended operations and controls (see 2.2.3.7, “Controls and Extended Operations” on page 41)
- A list of supported SASL security mechanisms
- A list of alternate LDAP servers

As LDAP is extended, additional information about the server will be stored in the root DSE.

2.2.3 The Functional Model

LDAP defines operations for accessing and modifying directory entries. This section discusses LDAP operations in a programming language-independent

manner. See Chapter 4, “Building LDAP-Enabled Applications” on page 85, for information on writing programs that invoke these operations.

LDAP operations can be divided into the following three categories:

Query Includes the search and compare operations used to retrieve information from a directory

Update Includes the add, delete, modify, and modify RDN operations used to update stored information in a directory

Authentication Includes the bind, unbind, and abandon operations used to connect and disconnect to and from an LDAP server, establish access rights and protect information

The most common operation is search. The search operation is very flexible and has some of the most complex options.

2.2.3.1 Search

The search operation allows a client to request that an LDAP server search through some portion of the DIT for information meeting user-specified criteria in order to read and list the result(s). There are no separate operations for read and list; they are incorporated in the search function. The search can be very general or very specific. The search operation allows one to specify the starting point within the DIT, how deep within the DIT to search, what attributes an entry must have to be considered a match, and what attributes to return for matched entries.

Some example searches expressed informally in English are:

- Find the postal address for `cn=John Smith,o=IBM,c=DE`.
- Find all the entries that are children of the entry `ou=ITSO,o=IBM,c=US`.
- Find the e-mail address and phone number of anyone in IBM whose last name contains the characters “miller” and who also has a fax number.

To perform a search, the following parameters must be specified (refer to Figure 12 on page 38):

- Base

  A DN that defines the starting point, called the base object, of the search. The base object is a node within the DIT.

- Scope

  Specifies how deep within the DIT to search from the base object. There are three choices: `baseObject`, `singleLevel`, and `wholeSubtree`. If `baseObject` is specified, only the base object is examined. If `singleLevel` is specified,

only the immediate children of the base object are examined; the base object itself is not examined. If `wholeSubtree` is specified, the base object and all of its descendants are examined.

- Search Filter

  Specifies the criteria an entry must match to be returned from a search. The search filter is a Boolean combination of attribute value assertions. An attribute value assertion tests the value of an attribute for equality, less than or equal, and so on. For example, a search filter might specify entries with a common name containing “wolf” or belonging to the organization ITSO. Search filters are discussed more fully in 2.2.3.3, “Search Filter Syntax” on page 39.

- Attributes to Return

  Specifies which attributes to retrieve from entries that match the search criteria. Since an entry may have many attributes, this allows the user to only see the attributes they are interested in. Normally, the user is interested in the value of the attributes. However, it is possible to return only the attribute types and not their values. This could be useful if a large value like a JPEG photograph was not needed for every entry returned from the search, but some of the photographs would be retrieved later as needed.

- Alias Dereferencing

  Specifies if aliases are dereferenced—that is, if the alias entry itself or the entry it points to is used. Aliases can be dereferenced or not when locating the base object and/or when searching under the base object. If aliases are dereferenced, then they are alternate names for objects of interest in the directory. Not dereferencing aliases allows the alias entries themselves to be examined.

- Limits

  Searches can be very general, examining large subtrees and causing many entries to be returned. The user can specify time and size limits to prevent wayward searching from consuming too many resources. The size limit restricts the number of entries returned from the search. The time limit limits the total time of the search. Servers are free to impose stricter limits than requested by the client.

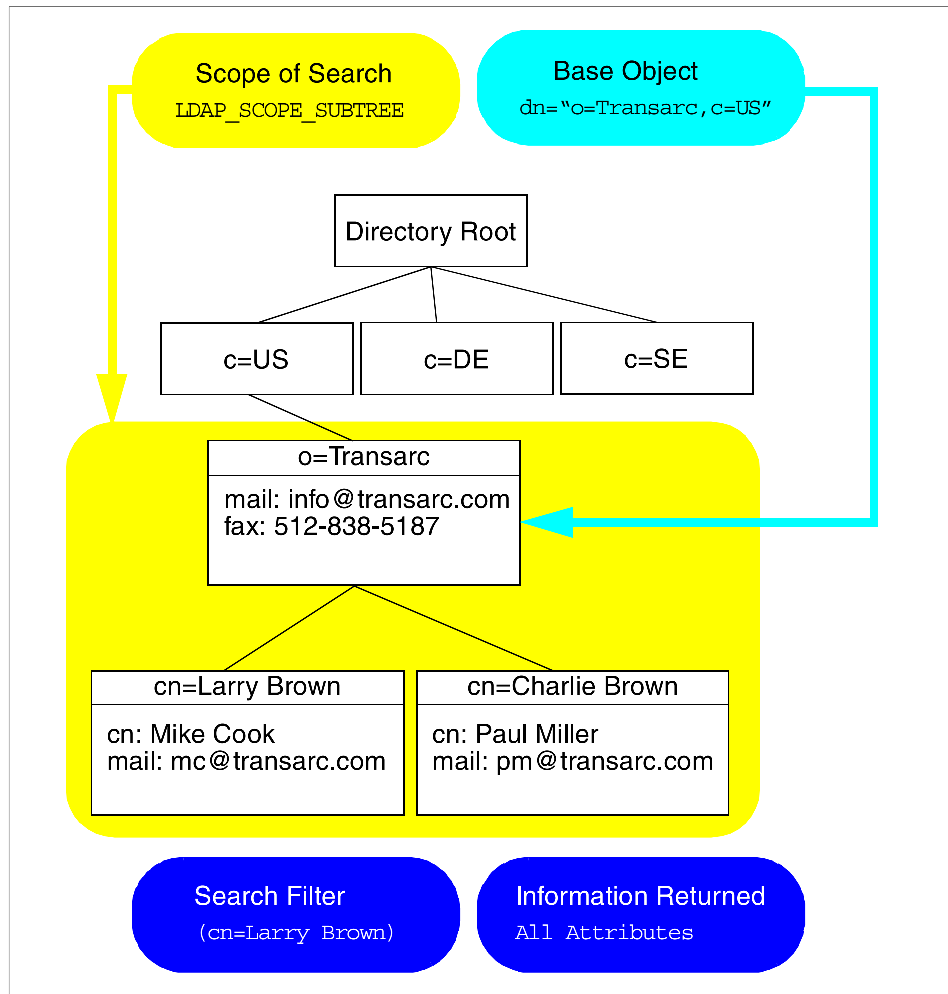


Figure 12. Search Parameters

2.2.3.2 Referrals and Continuation References

If the server does not contain the base object, it will return a referral to a server that does, if possible. Once the base object is found `singleLevel` and `wholeSubtree` searches may encounter other referrals. These referrals are returned in the search result along with other matching entries. These referrals are called continuation references because they indicate where a search could be continued.

For example, when searching a subtree for anybody named Smith, a continuation reference to another server might be returned, possibly along

with several other matching entries. It is not guaranteed that an entry for somebody named Smith actually exists at that server, only that the continuation reference points to a subtree that could contain such an entry. It is up to the client to follow continuation references if desired.

Since only LDAP Version 3 specifies referrals, continuation references are not supported in earlier versions.

2.2.3.3 Search Filter Syntax

The search filter defines criteria that an entry must match to be returned from a search. The basic component of a search filter is an attribute value assertion of the form:

```
attribute operator value
```

For example, to search for a person named John Smith the search filter would be `cn=John Smith`. In this case, `cn` is the attribute; `=` is the operator, and `John Smith` is the value. This search filter matches entries with the common name John Smith.

Table 6 lists the operators for search filters.

Table 6. Search Filter Operators

Operator	Description	Example
=	Returns entries whose attribute is equal to the value.	`cn=John Smith` finds the entry with common name John Smith
>=	Returns entries whose attribute is greater than or equal to the value.	`sn>=smith` finds all entries from smith to z*
<=	Returns entries whose attribute is less than or equal to the value.	`sn<=smith` finds all entries from `a*` to `smith`
=*	Returns entries that have a value set for that attribute.	`sn=*` finds all entries that have the `sn` attribute
~=	Returns entries whose attribute value approximately matches the specified value. Typically, this is an algorithm that matches words that sound alike.	`sn~= smit` might find the entry "`sn=smith`"

The "*" character matches any substring and can be used with the = operator. For example, `cn=J*Smi*` would match John Smith and Jan Smitty.

Search filters can be combined with Boolean operators to form more complex search filters. The syntax for combining search filters is:

```
( "&" or "|" (filter1) (filter2) (filter3) ...)
("!" (filter))
```

The Boolean operators are listed in Table 7.

Table 7. Boolean Operators

<table>
<tr><th>Boolean Operator</th><th>Description</th></tr>
<tr><td>&</td><td>Returns entries matching all specified filter criteria.</td></tr>
<tr><td>|</td><td>Returns entries matching one or more of the filter criteria.</td></tr>
<tr><td>!</td><td>Returns entries for which the filter is not true. This operator can only be applied to a single filter. <code>(!(filter))</code> is valid, but <code>(!(filter1)(filter2))</code> is not.</td></tr>
</table>

For example, `(|(sn=Smith)(sn=Miller))` matches entries with the surname Smith or the surname Miller. The Boolean operators can also be nested as in `(| (sn=Smith) (&(ou=Austin)(sn=Miller)))`, which matches any entry with the surname Smith or with the surname Miller that also has the organizational unit attribute Austin.

2.2.3.4 Compare

The compare operation compares an entry for an attribute value. If the entry has that value, compare returns TRUE. Otherwise, compare returns FALSE. Although compare is simpler than a search, it is almost the same as a base scope search with a search filter of attribute=value. The difference is that if the entry does not have the attribute at all (the attribute is not present), the search will return not found. This is indistinguishable from the case where the entry itself does not exist. On the other hand, compare will return FALSE. This indicates that the entry does exist, but does not have an attribute matching the value specified.

2.2.3.5 Update Operations

Update operations modify the contents of the directory. Table 8 summarizes the update operations.

Table 8. Update Operations

Operation	Description
add	Inserts new entries into the directory.
delete	Deletes existing entries from the directory. Only leaf nodes can be deleted. Aliases are not resolved when deleting.

Operation	Description
modify	Changes the attributes and values contained within an existing entry. Allows new attributes to be added and existing attributes to be deleted or modified.
modify DN	Change the least significant (left most) component of a DN or moves a subtree of entries to a new location in the DIT. Entries cannot be moved across server boundaries.

2.2.3.6 Authentication Operations

Authentication operations are used to establish and end a session between an LDAP client and an LDAP server. The session may be secured at various levels ranging from an insecure anonymous session, an authenticated session in which the client identifies itself by providing a password, to a secure, encrypted session using SASL mechanisms. SASL was added in LDAP Version 3 to overcome the weak authentication in LDAP Version 2 (some vendors, however, have added stronger authentication methods, such as Kerberos, to LDAP Version 2). Table 9 summarizes the authentication operations. The security aspects are discussed further in 2.2.4, “The Security Model” on page 42 and in 2.3, “Security” on page 43.

Table 9. Authentication Operations

Operation	Description
Bind	Initiates an LDAP session between a client and a server. Allows the client to prove its identity by authenticating itself to the server.
Unbind	Terminates a client/server session.
Abandon	Allows a client to request that the server abandon an outstanding operation.

2.2.3.7 Controls and Extended Operations

Controls and extended operations allow the LDAP protocol to be extended without changing the protocol itself. Controls modify the behavior of an operation, and extended operations add new operations to the LDAP protocol. The list of controls and extensions supported by an LDAP server can be obtained by examining the empty DN at that server (see 2.2.2.3, “Server Information” on page 35).

Controls can be defined to extend any operation. Controls are added to the end of the operation’s protocol message. They are supplied as parameters to functions in the API. In the future, standard controls might be defined in LDAP-related RFCs.

A control has a dotted decimal string object ID used to identify the control, an arbitrary control value that holds parameters for the control, and a criticality level. If the criticality level is TRUE, the server must honor the control or if the server does not support the control, reject the entire operation. If the criticality level is FALSE, a server that does not support the control must perform the operation as if there was no control specified.

For example, a control might extend the delete operation by causing an audit record of the deletion to be logged to a file specified by the control value information.

An extended operation allows an entirely new operation to be defined. The extended operation protocol message consists of a dotted decimal string object ID used to identify the extended operation and an arbitrary string of operation-specific data.

2.2.4 The Security Model

As previously described, the security model is based on the bind operation. There are several different bind operations possible, and thus the security mechanism applied is different as well. One possibility is when a client requesting access supplies a DN identifying itself along with a simple clear-text password. If no DN and password is declared, an anonymous session is assumed by the LDAP server. The use of clear text passwords is strongly discouraged when the underlying transport service cannot guarantee confidentiality and may therefore result in disclosure of the password to unauthorized parties.

Additionally, a Kerberos bind is possible in LDAP Version 2, but this has become deprecated in LDAP Version 3. Instead, LDAP V3 comes along with a bind command supporting the Simple Authentication and Security Layer (SASL) mechanism. This is a general authentication framework, where several different authentication methods are available for authenticating the client to the server; one of them is Kerberos. We discuss authentication in more detail in the following section 2.3, “Security” on page 43.

Furthermore, extended protocol operations are available in LDAP V3. An extension related to security is the “Extension for Transport Layer Security (TLS) for LDAPv3” which, at the time this book was written, is an Internet Draft (see A.4, “Other Sources” on page 140 for an URL). It defines operations that use TLS as a means to encrypt an LDAP session and protect it against spoofing. TLS is defined in “The TLS Protocol” Version 1.0, which is also still an Internet Draft. It is based on the Secure Socket Layer (SSL) Protocol 3.0, devised by Netscape Communications Corporation which it

eventually will supersede. TLS has a mechanism which enables it to communicate to an SSL server so that it is backwards compatible. The basic principles of SSL and TLS are the same and are further detailed in the following section 2.3, “Security” on page 43.

Some vendors, like Netscape and IBM, have already extended the LDAP protocol and added some SSL specific commands so that an encrypted TCP/IP connection is possible, thus providing a means for eliminating the need of sending a DN and a password unprotected over the network

Once a client is identified, access control information can be consulted to determine whether or not the client has sufficient access permissions to do what it is requesting.

2.3 Security

Security is of great importance in the networked world of computers, and this is true for LDAP as well. When sending data over insecure networks, internally or externally, sensitive information may need to be protected during transportation. There is also a need to know who is requesting the information and who is sending it. This is especially important when it comes to the update operations on a directory. The term security, as used in the context of this book, generally covers the following four aspects:

Authentication Assurance that the opposite party (machine or person) really is who he/she/it claims to be.

Integrity Assurance that the information that arrives is really the same as what was sent.

Confidentiality Protection of information disclosure by means of data encryption to those who are not intended to receive it.

Authorization Assurance that a party is really allowed to do what he/she/it is requesting to do. This is usually checked after user authentication. In LDAP Version 3, this is currently not part of the protocol specification and is therefore implementation- (or vendor-) specific. This is basically achieved by assigning access controls, like read, write, or delete, to user IDs or common names. There is an Internet Draft that proposes access control for LDAP.

The following sections focus on the first three aspects (since authorization is not contained in the LDAP Version 3 standard): authentication, integrity and confidentiality. There are several methods that can be used for this purpose; the most important ones are discussed here. These are:

- No authentication
- Basic authentication
- Simple Authentication and Security Layer (SASL)

Because no other data encryption method was available in LDAP Version 2, some vendors, for example Netscape and IBM, added their own SSL calls to the LDAP API. A potential drawback of such an approach is that the API calls might not be compatible among different vendor implementations. Therefore, in LDAP Version 3, a proposal is made (Extension for Transport Layer Security) to include SSL or, more accurately, its successor, TLS, through extended protocol operations. This should make the vendor-dependent functions redundant in the near future.

2.3.1 No Authentication

This is the simpliest way, one that obviously does not need to be explained in much detail. This method should only be used when data security is not an issue and when no special access control permissions are involved. This could be the case, for example, when your directory is an address book browsable by anybody. No authentication is assumed when you leave the password and DN field empty in the bind API call (see also Chapter 4, "Building LDAP-Enabled Applications" on page 85). The LDAP server then automatically assumes an anonymous user session and grants access with the appropriate access controls defined for this kind of access (not to be confused with the SASL anonymous user as discussed in 2.3.3, "Simple Authentication and Security Layer (SASL)" on page 45).

2.3.2 Basic Authentication

The security mechanism in LDAP is negotiated when the connection between the client and the server is established. This is the approach specified in the LDAP application program interface (API). Beside the option of using no authentication at all, the most simple security mechanism in LDAP is called basic authentication, which is also used in several other Web-related protocols, such as in HTTP.

When using basic authentication with LDAP, the client identifies itself to the server by means of a DN and a password which are sent in the clear over the network (some implementation may use Base64 encoding instead). The server considers the client authenticated if the DN and password sent by the client matches the password for that DN stored in the directory. Base64 encoding is defined in the Multipurpose Internet Mail Extensions (MIME)

standard (RFC 1521). It is a relatively simple encryption, and therefore it is not hard to break once one has captured the data on the network.

2.3.3 Simple Authentication and Security Layer (SASL)

SASL is a framework for adding additional authentication mechanisms to connection-oriented protocols. It has been added to LDAP Version 3 to overcome the authentication shortcomings of Version 2. SASL was originally devised to add stronger authentication to the IMAP protocol. SASL has since evolved into a more general system for mediating between protocols and authentication systems. It is a proposed Internet standard defined in RFC 2222.

In SASL, connection protocols, like LDAP, IMAP, and so on, are represented by profiles; each profile is considered a protocol extension that allows the protocol and SASL to work together. A complete list of SASL profiles can be obtained from the Information Sciences Institute (ISI). See A.4, “Other Sources” on page 140, for URL references. Among these are IMAP4, SMTP, POP3, and LDAP. Each protocol that intends to use SASL needs to be extended with a command to identify an authentication mechanism and to carry out an authentication exchange. Optionally, a security layer can be negotiated to encrypt the data after authentication and so ensure confidentiality. LDAP Version 3 includes such a command (`ldap_sasl_bind()`).

The SASL bind operation is explained in more detail with an example in 4.2.8, “Authentication Methods” on page 108. The key parameters that influence the security method used are:

dn — This is the distinguished name of the entry you want to bind as. This can be thought of as the user ID in a normal user ID and password authentication.

mechanism — This is the name of the security method that should be used. Valid security mechanisms are currently Kerberos Version 4, S/Key, GSSAPI, CRAM-MD5 and EXTERNAL. There is also an ANONYMOUS mechanism available which enables an authentication as user “anonymous”. In LDAP, the most common mechanism used is SSL (or its successor, TLS), which is provided as an EXTERNAL mechanism.

credentials — This contains the arbitrary data that identifies the DN. The format and content of the parameter depends on the mechanism chosen. If it is, for example, the ANONYMOUS mechanism, it can be an arbitrary string or an e-mail address that identifies the user.

Through the SASL bind API function call, LDAP client applications call the SASL protocol driver on the server, which in turn connects the authentication system named in the SASL mechanism to retrieve the required authentication information for the user. SASL can be seen as intermediator between the authentication system and a protocol like LDAP. Figure 13 illustrates this relationship.

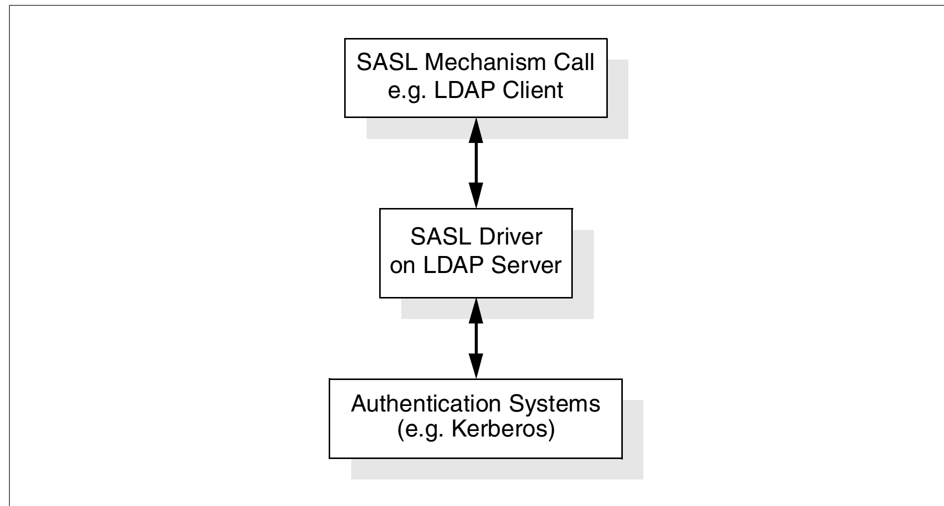


Figure 13. SASL Mechanism

Of course, the server must support this SASL mechanism as well, otherwise the authentication process will not be able to succeed. To retrieve a list of SASL mechanisms supported by an LDAP server (Version 3 only), point your Web browser to the following URL:

```
ldap://<ldap server>/?supportedsaslmechanisms
```

This is actually an LDAP URL, very similar to those used for HTTP (`http://<host>/...`) or other Internet protocols. You can get more information about LDAP URLs in 4.4, “LDAP URLs” on page 120.

As we have seen, the basic idea behind SASL is that it provides a high level framework that lets the involved parties decide on the particular security mechanism to use. The SASL security mechanism negotiation between client and server is done in the clear. Once the client and the server have agreed on a common mechanism, the connection is secure against modifying the authentication identities. An attacker could now try to eavesdrop the mechanism negotiation and cause a party to use the least secure mechanism. In order to prevent this from happening, clients and servers

should be configured to use a minimum security mechanism, provided they support such a configuration option.

As stated earlier, SSL and its successor, TLS, are the mechanisms commonly used in SASL for LDAP. Following is a brief description of SSL and TLS.

2.3.3.1 SSL and TLS

The Secure Socket Layer (SSL) protocol was devised to provide both authentication and data security. It encapsulates the TCP/IP socket so that basically every TCP/IP application can use it to secure its communication. See Figure 14.

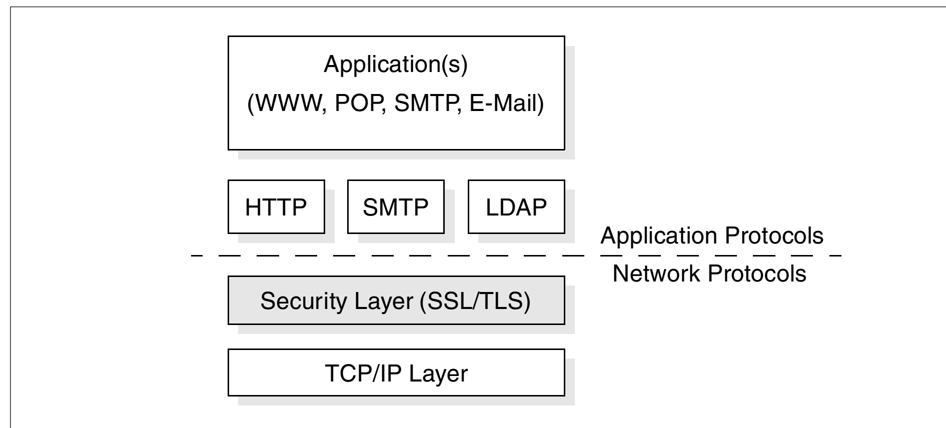


Figure 14. SSL/TLS in Relationship with Other Protocols

SSL was developed by Netscape and the current version is 3.0. Transport Layer Security (TLS) is an evolving open standard, currently in the state of an Internet Draft, being worked on at the IETF. It is based on SSL 3.0 with only a few minor differences, and it provides backwards compatibility with SSL 3.0. It is assumed that TLS will replace SSL. The following discussion is equally valid for both SSL and TLS.

SSL/TLS supports server authentication (client authenticates server), client authentication (server authenticates client), or mutual authentication. In addition, it provides for privacy by encrypting data sent over the network.

SSL/TLS uses a public key method to secure the communication and to authenticate the counterparts of the session. This is achieved with a public/private key pair. They operate as reverse functions to each other, which means data encrypted with the private key can be decrypted with the public key and vice versa. The assumption for the following considerations is that

the server has its key pair already generated. This is usually done when setting up the LDAP server.

The simplified interchange between a client and a server negotiating an SSL/TLS connection is explained in the following segment and illustrated in Figure 15.

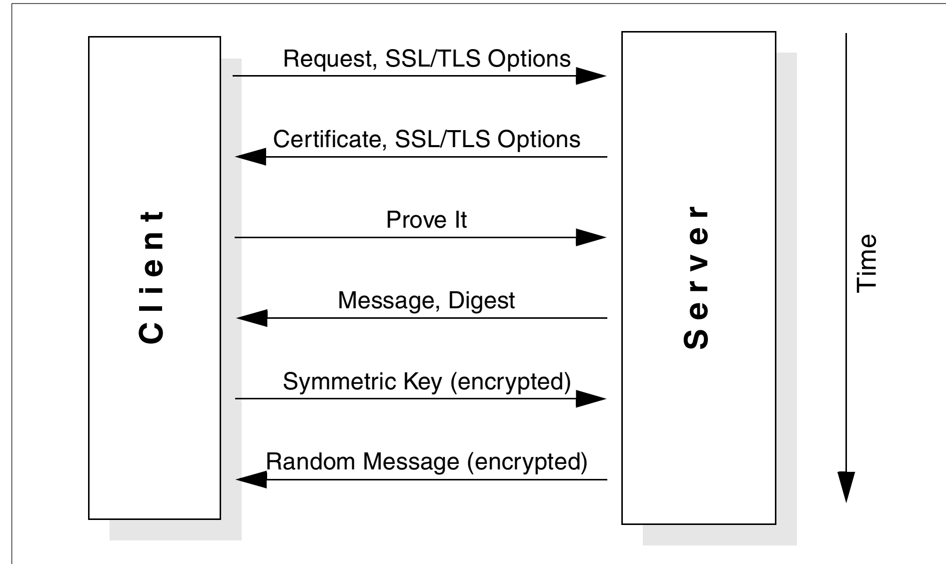


Figure 15. SSL/TLS Handshake

1. As a first step, the client asks the server for an SSL/TLS session. The client also includes the SSL/TLS options it supports in the request.
2. The server sends back its SSL/TLS options and a certificate which includes, among other things, the server's public key, the identity for whom the certificate was issued (as a distinguished name), the certifier's name and the validity time. A certificate can be thought of the electronic equivalent of a passport. It has to be issued by a general, trusted Certificate Authority (CA) which vouches that the public key really belongs to the entity mentioned in the certificate. The certificate is signed by the certifier which can be verified with the certifier's freely available public key
3. The client then requests the server to prove its identity. This is to make sure that the certificate was not sent by someone else who intercepted it on a former occasion.
4. The server sends back a message including a message digest (similar to a check sum) which is encrypted with its private key. A message digest that is computed from the message content using a hash function has two

features. It is extremely difficult to reverse, and it is nearly impossible to find a message that would produce the same digest. The client can decrypt the digest with the server's public key and then compare it with the digest it computes from the message. If both are equal, the server's identity is proved, and the authentication process is finished.

5. Next, server and client have to agree upon a secret (symmetric) key used for data encryption. Data encryption is done with a symmetric key algorithm because it is more efficient than the computing-intensive public key method. The client therefore generates a symmetric key, encrypts it with the server's public key, and sends it to the server. Only the server with its private key can decrypt the secret key.
6. The server decrypts the secret key and sends back a test message encrypted with the secret key to prove that the key has safely arrived. They can now start communicating using the symmetric key to encrypt the data.

As outlined above, SSL/TLS is used to authenticate a server to a client using its certificate and its private key and to negotiate a secret key later on used for data encryption. An example on how SSL can be used in a client application can be found in 4.2.8, "Authentication Methods" on page 108.

2.3.3.2 Other SASL Authentication Mechanisms

Although the SASL concepts supports multiple mechanisms, a particular vendor product may not support them all. It is very likely that vendor products only support a few mechanisms, such as SSL or TLS as just discussed above. Another common authentication method widely used is Kerberos. It has its roots in universities, where it proved to be scalable up to many thousands of clients. Kerberos is a third-party authentication method that uses a separate server providing security functions for the authentication process between involved parties. It uses the widely accepted Data Encryption Standard (DES) for message encryption.

2.4 Manageability

The LDAP specifications contained in the pertinent RFCs (as listed and briefly explained in 2.1, "Overview of LDAP Architecture" on page 19) include functions for directory data management. These include functions to create and modify the directory information tree (DIT) and to add, modify, and delete data stored in the directory.