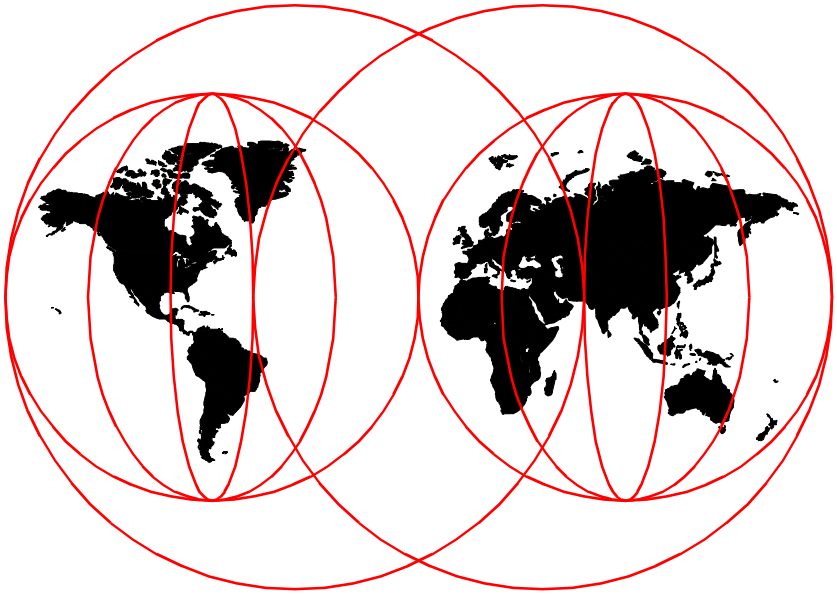
LDAP Implementation Cookbook

Heinz Johner, Michel Melot, Harri Stranden, Permana Widhiasta



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5110-00

SG24-5110-00

International Technical Support Organization

LDAP Implementation Cookbook

June 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 273.

First Edition (June 1999)

This edition applies to the IBM SecureWay Directory, available for IBM AIX, IBM OS/400, IBM OS/390, Microsoft Windows NT, and Sun Solaris.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B, Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

The implementation and exploitation of centralized, corporate-wide directories are among the top priority projects in most organizations in the years 1999 and 2000. The need for a centralized directory emerges as organizations realize the overhead and cost involved in managing the many distributed micro and macro directories introduced in the past decade with decentralized client/server applications and network operating systems.

Directories are key for a successful IT operation in medium and large environments. IBM understands this requirement and supports it by providing directory implementations based on industry standards at no additional cost on all its major platforms and even important non-IBM platforms. The *IBM SecureWay Directory*, formerly known as the *IBM eNetwork LDAP Directory*, implements the Lightweight Directory Access Protocol (LDAP) standard that has emerged quickly in the past years as a result of the demand for such a standard.

This redbook will help you understand, install, and configure the IBM SecureWay Directory. It is targeted at system specialists who need to know the concepts and the detailed instructions for a successful LDAP implementation.

The Team That Wrote This Redbook

This redbook is an evolution of an IBM working paper, *IBM eNetwork LDAP Exploitation Guide*, that was written by **Mike Schlosser** and **Ellen Stokes** at IBM Austin. The redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Heinz Johner is an Advisory I/T Specialist at the International Technical Support Organization, Austin Center. He writes extensively on the Distributed Computing Environment (DCE), security-related areas, and eNetwork deployment. Before joining the ITSO, he worked in the Professional Services organization of IBM Switzerland and was responsible for DCE and Systems Management in medium and large customer projects.

Michel Melot is an I/T Specialist from the IBM Service Delivery in La Gaude, France. He has many years of experience in the computer industry and specializes in DCE/DFS implementation and administration for the European environment.

Harri Stranden is an I/T Architect for e-business Solutions at IBM Finland. He has over 20 years of experience in the IT field on networking and Systems Management. He has worked with IBM for 12 years. His areas of expertise include building solutions exploiting Internet technologies.

Permana Widhiasta is an I/TAP System Services Representatives from IBM Global Services in Indonesia. He has worked with IBM for several years. He holds a degree in Materials Engineering from the University of Indonesia. His area of expertise includes AS/400 Systems Management.

Thanks to the following people for their invaluable contributions to this book:

Matt Barber, IBM Austin
Debora Byrne, IBM Austin
Patrick Fleming, IBM Rochester
Mike Garrison, IBM Austin
Karen Gdaniec, IBM Endicott
Timothy Hahn, IBM Endicott
Kyle L. Henderson, IBM Rochester
Paul Immanuel, IBM Raleigh
James Manon, IBM Austin
Mark McConaughy, IBM Austin
Ellen Stokes, IBM Austin
Diane H. Pearce, IBM Canada

Paul de Graaff, ITSO Poughkeepsie
Fant Steele, ITSO Rochester

Special thanks go to the editors for their invaluable help in finalizing the book:

John Owczarzak
Milos Radosavlijevic

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 293 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Introduction

People and businesses are increasingly relying on networked computer systems to support distributed applications. These distributed applications might interact with computers on the same local area network (LAN), within a corporate intranet, within extranets linking up partners and suppliers, or anywhere on the worldwide Internet. To improve functionality and ease-of-use, and to enable cost-effective administration of distributed applications, information about the services, resources, users, and other objects accessible from the applications needs to be organized in a clear and consistent manner. Much of this information can be shared among many applications, but, it must also be protected in order to prevent unauthorized modification or the disclosure of private information.

Information describing the various users, applications, files, printers, and other resources accessible from a network is often collected into a special database that is sometimes called a directory. As the number of different networks and applications has grown, the number of specialized directories of information has also grown resulting in islands of information that are difficult to share and manage. If all of this information could be maintained and accessed in a consistent and controlled manner, it would provide a focal point for integrating a distributed environment into a consistent and seamless system.

The Lightweight Directory Access Protocol (LDAP) is an open industry standard that has evolved to meet these needs. LDAP defines a standard method for accessing and updating information in a directory. LDAP is gaining wide acceptance as the directory access method of the Internet and is therefore also becoming strategic within corporate intranets. It is being supported by a growing number of software vendors and is being incorporated into a growing number of applications. For example, the two most popular Web browsers, Netscape Navigator/Communicator and Microsoft Internet Explorer, support LDAP functionality as a base feature.

This *LDAP Implementation Cookbook* explains the steps to get started with LDAP directory implementation projects and is intended to give the reader a detailed understanding of the architecture, use, and benefits of LDAP. It also contains product-specific installation, configuration, and administration information.

Chapter 1 provides background information about what a directory service is and the benefits it can provide and what IBM offerings are. The schema and namespace are discussed in detail in Chapter 2. Chapter 3 proposes a

step-by-step approach for directory implementation projects. The management and administration of a centralized or distributed directory from a design and planning point of view are covered in Chapter 4. Next, Chapter 5 discusses security topics like authentication of directory users and access control lists (ACL). Chapter 6 provides information about the installation and configuration of the IBM SecureWay Directory. Directory and system administration considerations are discussed in Chapter 7. Chapter 8 finally explains how to develop directory-enabled applications. Various reference material is collected in the appendices.

1.1 What Is a Directory?

A directory is a collection of information about objects arranged in some order that gives details about each object. Popular examples are a city telephone directory and a library card catalog. For a telephone directory, the objects listed are people; the names are arranged alphabetically, and the details given about each person typically include an address and telephone number. Books in a library card catalog are ordered by author or by title, and information such as the ISBN number of the book and other publication information is also contained.

In computer terms, a directory is a specialized database, also called a data repository, that stores typed and ordered information about objects. A particular directory might list information about printers (the objects) consisting of typed information, such as location (a formatted character string), speed in pages per minute (numeric), print data streams supported (for example PostScript or ASCII), and so on.

Directories allow users or applications to find resources that have the characteristics needed for a particular task. For example, a directory of users can be used to look up a person's e-mail address or fax number. A directory can be searched to find a nearby PostScript color printer. Or, a directory of application servers can be searched to find a server that can access customer billing information.

The terms *white pages* and *yellow pages* are sometimes used to describe how a directory is used. If the name of an object (person, printer) is known, its characteristics (phone number and pages per minute) can be retrieved. This is similar to looking up a name in the white pages of a telephone directory. If the name of a particular individual object is not known, the directory can be searched for a list of objects that meet a certain requirement. This is like looking up a listing of hairdressers in the yellow pages of a telephone directory. However, directories stored on a computer are much more flexible

than the yellow pages of a telephone directory because they can usually be searched by specific criteria not just by a predefined set of categories.

1.1.1 Differences Between Directories and Databases

A directory is often described as a database, but it is a specialized database that has characteristics that set it apart from general purpose relational databases. One special characteristic of directories is that they are accessed (read or searched) much more often than they are updated (written). Hundreds of people might look up an individual's phone number, or thousands of print clients might look up the characteristics of a particular printer. But, the phone number or printer characteristics rarely ever change.

Because directories must be able to support high volumes of read requests, they are typically optimized for read access. Write access might be limited to system administrators or to the owner of each piece of information. A general purpose database, on the other hand, needs to support applications, such as airline reservations and banking applications, with relatively high update volumes.

Because directories are meant to store relatively static information and are optimized for that purpose, they are not appropriate for storing information that changes rapidly. For example, the number of jobs currently in a print queue probably should not be stored in the directory entry for a printer because that information would have to be updated frequently to be accurate. Instead, the directory entry for the printer can contain the network address of a print server. The print server can be queried to get the current queue length if desired. The information in the directory (the print server address) is static, whereas the number of jobs in the print queue is dynamic.

Another difference between directories and general purpose databases is that directories normally do not support transactions. This may, however, change in the future and some vendor implementation may already support transactions. Transactions are all-or-nothing operations that must be completed in total or not at all. For example, when transferring money from one bank account to another, the money must be debited from one account and credited to the other account in a single transaction. If only half of this transaction completes or someone accesses the accounts while the money is in transit, the accounts will not balance. General-purpose databases usually support such transactions, which complicates their implementation.

Because directories deal mostly with read requests, the complexities of transactions can be avoided. If two people exchange offices, both of their directory entries need to be updated with new phone numbers, office

locations, and so on. If one directory entry is updated, and the other directory entry is then updated, there is a brief period during which the directory will show that both people have the same phone number. Because updates are relatively rare, such anomalies are considered acceptable.

The type of information stored in a directory usually does not require strict consistency. It might be acceptable if information such as a telephone number is temporarily out of date. Because directories are (currently) not transactional, it is not a good idea to use them to store information that is sensitive to inconsistencies, such as bank account balances.

Because general-purpose databases must support arbitrary applications such as banking and inventory control, they allow arbitrary collections of data to be stored. Directories may be limited in the type of data they allow to be stored (although the architecture does not impose such a limitation). For example, a directory specialized for customer contact information might be limited to storing only personal information such as names, addresses, and phone numbers. If a directory is extensible, it can be configured to store a variety of types of information making it more useful to a variety of programs.

Another important difference between a directory and a general-purpose database is in the way information can be accessed. Most databases support a standardized, very powerful access method called Structured Query Language (SQL). SQL allows complex update and query functions at the cost of program size and application complexity. LDAP directories, on the other hand, use a simplified and optimized access protocol that can be used in slim and relatively simple applications.

Because directories are not intended to provide as many functions as general-purpose databases, they can be optimized to economically provide more applications with rapid access to directory data in large distributed environments. Because the intended use of directories is restricted to a read-mostly, non transactional environment, both the directory client and directory server can be simplified and optimized.

What About the Future?

Many of the differences just mentioned may lead to the suspicion that a directory is no more than a limited-function database. This is indeed partly true since one of the important design goals of a directory service is that it can be accessed and used from relatively small and simple applications. In fact, certain vendor products, such as the IBM SecureWay Directory, use a relational database under the cover to implement the functions. Also, proposals are being discussed in the standards bodies to add some functions to future versions of LDAP that currently are specific to databases, such as support for transactional updates.

1.1.2 Directory Clients and Servers

Directories are usually accessed using the client/server model of communication. An application that wants to read or write information in a directory does not access the directory directly. Instead, it calls a function or application programming interface (API) that causes a message to be sent to another process. This second process accesses the information in the directory on behalf of the requesting application. The results of the read or write are then returned to the requesting application (see Figure 1).

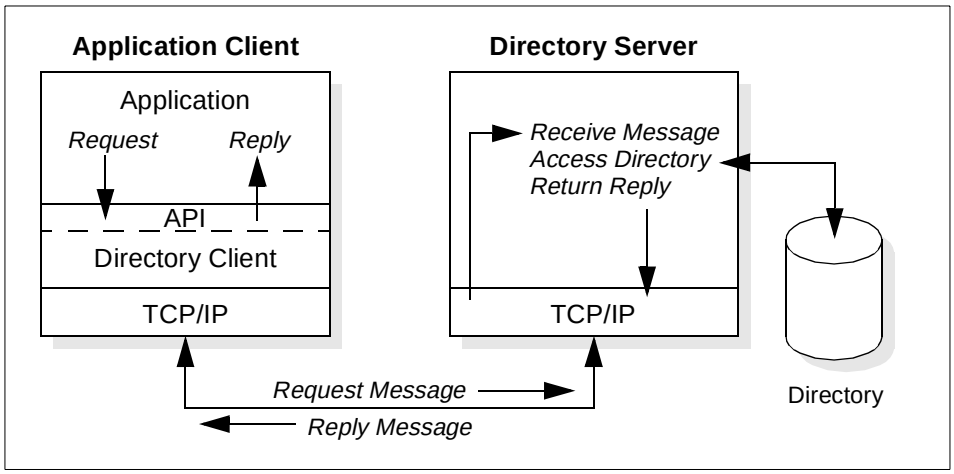


Figure 1. Directory client/server interaction

The request is performed by the directory client, and the process that maintains and looks up information in the directory is called the directory server. In general, servers provide a specific service to clients. Sometimes, a

server might become the client of other servers in order to gather the information necessary to process a request.

A directory service is only one type of service that might be available in a client/server environment. Other common examples of services are file services, mail services, print services, Web page services, and so on. The client and server processes might or might not be on the same machine. A server is capable of serving many clients. Some servers can process client requests in parallel. Other servers queue incoming client requests for serial processing if they are currently busy processing another client's request.

An API defines the programming interface that a particular programming language uses to access a service. The format and contents of the messages exchanged between client and server must adhere to an agreed upon protocol. LDAP defines a message protocol used by directory clients and directory servers. There is also an associated LDAP API for the C language and ways to access the directory from a Java application using JNDI (see Chapter 8, "Developing Directory-Enabled Applications" on page 223 for more details on these APIs). The client is not dependent upon a particular implementation of the server, and the server can implement the directory however it chooses.

It may seem confusing, at this time, to discuss directories as well as protocols while the acronym *LDAP* strictly refers to a protocol only. Section 1.3, "LDAP: Protocol or Directory?" on page 13 will clarify this.

1.1.3 Distributed Directories

The terms *local*, *global*, *centralized*, and *distributed* are often used to describe a directory or directory service. These terms mean different things to different people in different contexts. In this section, these terms are explained as they apply to directories in different contexts.

In general, *local* means something is close by, and *global* means that something is spread across the universe of interest. The universe of interest might be a company, a country, or the Earth. Local and global are two ends of a continuum. That is, something may be more or less global or local than something else. Centralized means that something is in one place, and distributed means that something is in more than one place. Like local and global, something can be distributed to a greater or lesser extent.

The information stored in a directory can be local or global in scope. For example, a directory that stores local information might consist of the names, e-mail addresses, public encryption keys, and so on of members of a

department or workgroup. A directory that stores global information might store information for an entire company. Here, the universe of interest is the company.

The clients that access information in the directory can be local or global. Local clients may all be located in the same building or on the same LAN. Global clients might be distributed across the continent or planet.

The directory itself can be *centralized* or *distributed*. If a directory is centralized, there is one directory server (or a server cluster) at one location that provides access to the directory. If the directory is distributed, there are multiple servers, usually geographically dispersed, that provide access to the directory.

When a directory is distributed, the information stored in the directory can be *partitioned* or *replicated*. When information is partitioned, each directory server stores a unique and non-overlapping subset of the information. That is, each directory entry is stored by one and only one server. The technique to partition the directory is to use LDAP referrals. LDAP referrals allow the users to refer LDAP requests to either the same or different name spaces stored in a different (or same) server. When information is replicated, the same directory entry is stored by more than one server. In a distributed directory, some information may be partitioned, and some information may be replicated.

The three *dimensions* of a directory — scope of information, location of clients, and distribution of servers — are independent of each other. For example, clients scattered across the globe can access a directory containing only information about a single department, and that directory can be replicated at many directory servers. Or, clients in a single location can access a directory containing information about everybody in the world that is stored by a single directory server.

The scope of information to be stored in a directory is often given as an application requirement. The distribution of directory servers and the way in which data is partitioned or replicated can often be controlled to effect the performance and availability of the directory. For example, a distributed and replicated directory might perform better because a read request can be serviced by a nearby server. A centralized directory may be less available because it is a single point of failure. However, a distributed directory might be more difficult to maintain because multiple sites, possibly under the control of multiple administrators, must be kept up-to-date and in running order.

The design and maintenance of a directory service can be complex, and many trade-offs may be involved. You can see more detailed information about the topic in the following chapters.

1.1.4 Directory Security

The security of information stored in a directory is a major consideration. Some directories are meant to be accessed publicly on the Internet, but any user should not necessarily be able to perform any operation. A company's directory servicing its intranet can be stored behind a firewall to keep the general public from accessing it, but more security control may be needed within the intranet itself.

For example, anybody should be able to look up an employee's e-mail address, but only the employee or a system administrator should be able to change it. Members of the personnel department might have permission to look up an employee's home telephone number, but their co-workers might not. Depending on the confidentiality of the data, information may need to be encrypted before being transmitted over the network. A security policy defines who has what type of access to what information. The security policy is defined by the organization that maintains the directory.

A directory should support the basic capabilities needed to implement a security policy. The directory might not directly provide the underlying security capabilities, but it might be integrated with a trusted network security service that provides the basic security services. First, a method is needed to authenticate users. Authentication verifies that users are who they say they are. A user name and password is a basic authentication scheme. Once users are authenticated, it must be determined if they have the authorization or permission to perform the requested operation on the specific object.

Authorization is often based on access control lists (ACLs). An ACL is a list of authorizations that may be attached to objects and attributes in the directory. An ACL lists what type of access each user or a group of users is allowed or denied. In order to make ACLs shorter and more manageable, users with the same access rights are often put into security groups. Table 1 shows an example ACL for an employee's directory entry.

Table 1. Example ACL for an employee's directory entry

User or Group	Access Rights
owner	read, modify (but not delete)
administrators	all
personnel	read all fields

User or Group	Access Rights
all others	read restricted

The authentication and authorization methods and related standards and proposals relevant to LDAP, like *Secure Sockets Layer (SSL)* and *Simple Authentication and Security Layer (SASL)*, are discussed in more detail in Chapter 5, “Directory Security” on page 103.

1.1.5 Users, Platforms, and Networks

The concept of *users*, what a user is, where the information about a user is stored, how it is used, and so on is key to understanding the relationship between LDAP directory services and platform operating systems.

Starting with LDAP directory service, there are two obvious ways to think about users:

- A user may be an LDAP directory user. LDAP sessions begin with a bind operation. For cases other than unauthenticated access, a bind request must supply a distinguished name (DN) and a password or another type of credential. So, a user must be defined in the directory server (the user must have a DN) in order to be authenticated to access the directory.
- A user may be a white pages entry in an LDAP directory. The common white pages or phone book application will contain information about people, in most cases many people. There can be, but does not have to be, a relationship between a white pages entry and an LDAP directory user; that is, everyone listed in the white pages directory may not have access to the directory. The inverse may also be true; people not listed in the directory may access the data as, for example, when internal users access a directory that stores customer person objects.

Most operating system platforms provide local or remote access for users; so, a user can also be the following:

- A user may be a system-defined user. The user must be defined on the system platform in order to be authenticated so he/she can be a user of that platform for access to system services, such as file and print sharing. Such users are usually defined by a user ID and a password, or, with increasing importance, by a digital certificate signed by a known Certificate Authority (CA).

Historically, multi-user interactive applications and subsystems used the platform-level user definitions on the hosted platform to authenticate user access to transactions, programs and data. While single sign-on, and the

overall security architecture are beyond the scope of this guide, it is sufficient to say that, as the number of systems in a network continues to grow and applications themselves are distributed across multiple systems, the strict application-to-host system user affinity is insufficient. So, many applications have developed their own *private* user and authentication files.

- A user may be an application user. The user must be defined/enrolled in the specific application in order to be authenticated and have access to that application and its resources.

It is important to understand this current topology with its mix of semantics, syntax, and backing stores for *user* information in order to understand the context of *users* within a distributed environment.

1.2 The Directory as Infrastructure

A directory that is accessible by multiple applications is a vital part of the infrastructure supporting networked systems. A directory service provides a single logical view of the users, resources, and other objects that make up a distributed system. This allows users and applications to access network resources transparently. That is, the system is perceived as an integrated whole, a network, not a collection of independent parts. Objects can be accessed by name or function without knowing low-level identifiers such as host addresses, file server names, and e-mail addresses.

1.2.1 Directory-Enabled Applications

A directory-enabled application is an application that uses a directory service to improve its functionality, ease of use, and administration. Today, many applications make use of information that can be stored in a directory. For example, consider a group calendar application that is used to schedule meetings of company personnel in different conference rooms.

In the worst case, the calendar application does not use a directory service at all. If this were the case, a user trying to schedule a meeting would have to remember the room number of every conference room that might be appropriate for the meeting. Is the room big enough, does it have the necessary audio and video equipment, and so on? The user would also have to remember the names and e-mail addresses of every attendee that needs to receive a meeting notice. Such an application would obviously be cumbersome to use.

If conference room information (size, location, special equipment, and so on) and personnel information (name, e-mail address, phone number, and so on)

can be accessed from a directory service, the application will be much easier to use. Also, the functionality of the application can be improved. For example, a list of all available conference rooms meeting the size and equipment requirements can be presented to the user.

But, the developers of directory-enabled applications are faced with a problem. What if they cannot assume that a directory service will exist in all environments? If there is a directory service, it might be specific to a certain network operating system (NOS) making the application non-portable. Can the existing directory service be extended to store the type of information needed by the application? Because of these concerns, application developers often take the approach of developing their own application-specific directory.

1.2.2 The Benefits of a Common Directory for Applications

An application-specific directory stores only the information needed by a particular application and is not accessible by other applications. Because a full-function directory service is complex to build, application-specific directories are typically very limited. They probably store only a specific type of information, probably do not have general search capabilities, probably do not support replication and partitioning, and probably do not have a full set of administration tools. An application-specific directory can be as simple as a set of editable text files, or it can be stored and accessed in an undocumented proprietary manner.

In such an environment, each application creates and manages its own application-specific directory, which quickly becomes an administrative nightmare. The same e-mail address stored by the calendar application might also be stored by a mail application and by an application that notifies system operators of equipment problems. Keeping multiple copies of information up-to-date and synchronized is difficult especially when different user interfaces and even different system administrators are involved.

What is needed is a common, application-independent directory. If application developers could be assured of the existence of a directory service, application-specific directories would not be necessary. However, a common directory must address the problems mentioned above. It must be based on an open standard that is supported by many vendors on many platforms. It must be accessible through a standard API. It must be extensible so that it can hold the types of data needed by arbitrary applications. And it must provide rich functionality without requiring excessive resources on smaller systems. Since more users and applications will access and depend on the common directory, it must also be robust, secure, and scalable.

When such a directory infrastructure is in place, application developers can devote their time to developing applications instead of application-specific directories. In the same way that developers rely on the communications infrastructure of TCP/IP, remote procedure call (RPC), and object request brokers (ORBs) to free them from low-level communication issues, they will be able to rely on powerful, full-function directory services. LDAP is the protocol to be used to access this common directory infrastructure. Like HTTP (hypertext transfer protocol) and FTP (file transfer protocol), LDAP will become an indispensable part of the Internet's protocol suite.

When applications access a standard common directory that is designed in a proper way, rather than using application-specific directories, redundant and costly administration can be eliminated, and security risks are more controllable. The calendar, mail, and operator notification applications can all access the same directory to retrieve an e-mail address. New uses for directory information will be realized, and a synergy will develop as more applications take advantage of the common directory.

1.2.3 Directory-Enabled Networks

But, it is not only directory-enabling your applications, but also directory-enabling all your network resources, such as users, systems, and network devices, and integrating them into the directory infrastructure. The information model and schema for the next generation of distributed networks has been defined by the Directory-Enabled Networks (DEN) initiative incorporated by Desktop Management Task Force's Common Information Model (CIM). The information models and schemas are discussed in more detail in Chapter 2, "Schema and Namespace" on page 31.

What does this mean to directory development projects? It means that the IBM SecureWay Directory or other LDAP-based directory services will define users and other resources. While this will occur in stages, the goal is to sign on to the network via the LDAP directory, rather than signing on to a platform or to an application. As the mechanisms for supporting the platform/system resource user and the application user decline, they will be replaced by the network-enabled user by providing the directory infrastructure for authentication and authorization of the network resources.

The guidelines for directory exploitation projects are:

- For new applications, use the IBM SecureWay Directory or other LDAP directory services for enrolling and defining users and resources and for authentication.

- For existing applications and subsystems: When re-engineering for network computing, implement LDAP based resource enrollment, definition, and authentication.
- For existing platform and platform services: When re-engineering for network computing or other initiatives, implement LDAP-based user and resource enrollment, definition, and authentication.

When it comes to administration of network resources, directory-enabled networks will bring even greater benefits than are described in 1.2.2, “The Benefits of a Common Directory for Applications” on page 11. Directory services are the central pillar for network computing.

1.3 LDAP: Protocol or Directory?

LDAP defines a communication protocol. That is, it defines the transport and format of messages used by a client to access data in an X.500-like directory. LDAP does not define the directory service itself. Yet, people often talk about LDAP directories. Others say LDAP is only a protocol and that there is no such thing as an LDAP directory. What is an LDAP directory? Let us first consider some basic information about X.500.

1.3.1 X.500: The Directory Service Standard

The CCITT (Comite Consultatif International Telephonique et Telegraphique or Consultative Committee on International Telephony and Telegraphy, which is nowadays ITU-T, International Telecommunications Union - Telecommunication Standardization Sector) defined the X.500 standard in 1988, which then became *ISO 9594, Data Communications Network Directory, Recommendations X.500/X.521* in 1990, though it is still commonly referred to as X.500.

X.500 organizes directory entries in a hierarchical name space capable of supporting large amounts of information. It also defines powerful search capabilities to make information retrieval easier. Because of its functionality and scalability, X.500 is often used together with add-on modules for interoperation between incompatible directory services.

X.500 specifies that communication between the directory client and the directory server uses the directory access protocol (DAP). However, as an application layer protocol, the DAP requires the entire OSI protocol stack to operate. Supporting the OSI stack requires more resources than are available in many small environments. Therefore, an interface to an X.500 directory server using a less resource-intensive or lightweight protocol was desired.

1.3.2 LDAP Server as a Gateway

An application client program initiates an LDAP message by calling an LDAP API. But, an X.500 directory server does not understand LDAP messages. In fact, the LDAP client and X.500 server even use different communication protocols (TCP/IP vs. OSI). The LDAP client actually communicates with a gateway process (also called a proxy or front end) that translates and forwards requests to the X.500 directory server (see Figure 2). This gateway is known as an LDAP server. It services requests from the LDAP client. It does this by becoming a client of the X.500 server. The LDAP server must communicate using both TCP/IP and OSI.

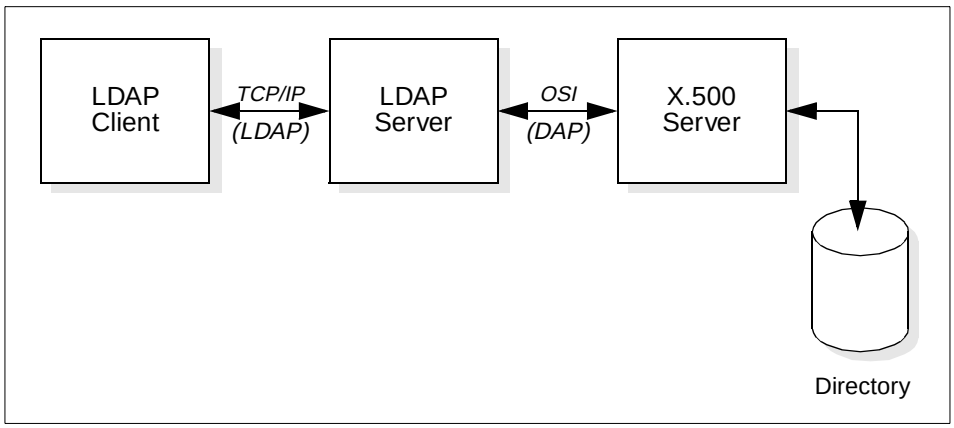


Figure 2. LDAP server acting as a gateway to an X.500 server

This way, clients can access the X.500 directory without dealing with the overhead and complexity which X.500 requires.

1.3.3 LDAP as Standalone Server

As the use of LDAP grew and its benefits became apparent, people who did not have X.500 servers or the environments to support them wanted to build directories that could be accessed by LDAP clients. So, why not have the LDAP server store and access the directory itself instead of only acting as a gateway to X.500 servers (see Figure 3). This eliminates any need for the OSI protocol stack. Of course, this makes the LDAP server much more complicated since it must now be able to store and retrieve directory entries. These LDAP servers are often called stand-alone LDAP servers because they do not depend on an X.500 directory server. Since LDAP does not support all X.500 capabilities, a stand-alone LDAP server only needs to support the capabilities required by LDAP.

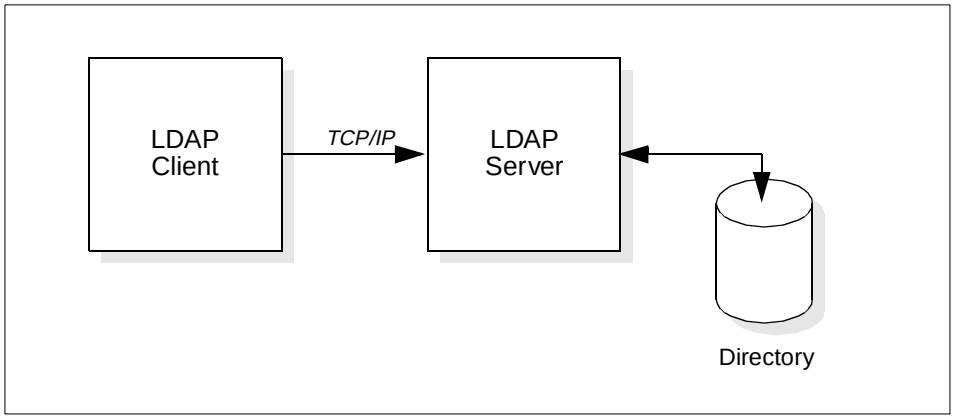


Figure 3. Stand-alone LDAP server

RFC 1777 (LDAP Version 2) discusses providing access to the X.500 directory. RFC 2251 (LDAP Version 3) discusses providing access to directories supporting the X.500 model. This change in language between Version 2 and Version 3 of the LDAP protocol standard reflects the idea that an LDAP server can implement the directory itself or can be a gateway to an X.500 directory.

From the client's point of view, any server that implements the LDAP protocol is an LDAP directory server, whether the server actually implements the directory or is a gateway to an X.500 server. The directory that is accessed can be called an LDAP directory whether the directory is implemented by a stand-alone LDAP server or by an X.500 server.

In fact, when referring to an LDAP server, as is the case throughout this book in reference to the IBM SecureWay Directory, only a standalone LDAP server is meant.

1.4 The LDAP Road Map

LDAP has evolved to meet the need of providing access to a common directory infrastructure. LDAP is an open industry standard that is supported by many system vendors on a variety of platforms. It is being incorporated into software products and is quickly becoming the directory access protocol of choice. LDAP allows products from different vendors on different platforms to interoperate and provide a global directory infrastructure much like HTTP enabled the deployment of the World Wide Web.

Current LDAP products support at least the LDAP Version 2 protocol level. Many products already support parts or all of LDAP Version 3. Further enhancements beyond Version 3 are being discussed by the IETF (Internet Engineering Task Force).

Application developers can take advantage of LDAP to develop next-generation directory-enabled applications. While X.500 has traditionally been deployed only in large organizations that can commit the resources necessary to support it, LDAP is also appropriate for small organizations. For example, a small company might want to exchange documents with its customers and suppliers using Electronic Data Interchange (EDI). EDI requires both parties to agree on the types of documents to be exchanged, communication requirements, and so on. Companies can publish their EDI characteristics in publicly accessible LDAP directories to facilitate EDI.

A common directory infrastructure encourages new uses. The Directory Enabled Networks (DEN) specification, for example, allows information about network configuration, protocol information, router characteristics, and so on to be stored in an LDAP directory. The availability of this information in a standardized format from many equipment vendors will allow for the intelligent management and provisioning of network resources. These examples show the diverse uses of directory-enabled applications supported by a common directory infrastructure accessed with LDAP.

1.4.1 LDAP Is More than a Programming Model

Most of the books and articles on LDAP provide sample applications that help to explain the LDAP programming model. These samples are quite often some form of a white pages (address book) application and are implemented on a pristine system as the sole user of the directory service.

These samples are valuable as a quick-start lesson for LDAP programming, but, while there are a few brand new applications written from scratch that will use an LDAP directory service, most directory exploitation projects will involve the re-engineering of existing applications. The LDAP literature does not provide much guidance for this complex re-engineering task.

The opportunity to implement an LDAP exploitation project on a pristine system or as the sole user of the directory service is similarly unlikely. Typically, there will be many applications using the directory accessing and sharing the data it contains. This, after all, is the primary objective driving the acceptance of a common access method and directory service.

In fact, very little is written about the real-life considerations that most directory exploitation projects experience. Based on the trivial sample application model, many projects set very aggressive schedules and design points and begin to code as if a new application using LDAP on a new directory service will be the only application. Exploiters in these projects quickly realize the need for application re-engineering and data modeling tasks which are, as designers and developers discover, fundamental to the success of directory exploitation projects.

It is the intention of his book to define some of the important design considerations for many of the directory exploitation and implementation projects. While not every topic may apply to every project, the understanding gained should help to make design goals and schedules more realistic and such projects more successful.

1.5 The Framework for Creating Enterprise-Wide Solutions

As businesses incorporate Internet technologies into their core business processes, they start to achieve real business value. Today, companies large and small are using the Web to communicate with their partners, to connect with their back-end data systems, and to conduct *electronic* commerce. This is *e-business* where the strength and reliability of traditional information technology meet the Internet. The IBM Application Framework for e-business (Figure 4 on page 18) is designed to enable businesses to build and deploy applications based on Internet and Web technologies quickly and easily. To accomplish this goal, the architecture is based on open industry standards (like LDAP among others) that are or have been widely adopted in the computer industry.

The Application Framework for e-business network infrastructure provides a platform for the entire e-business environment and includes TCP/IP and network services, security services, directory services, mobile services, client management services, and file and print services. The supported standards include LDAP, TCP/IP, Common Data Security Architecture (CDSA), Secure Socket Layer (SSL), and X.509v3 certificates among others. Directory services is a major component of the framework's network infrastructure.

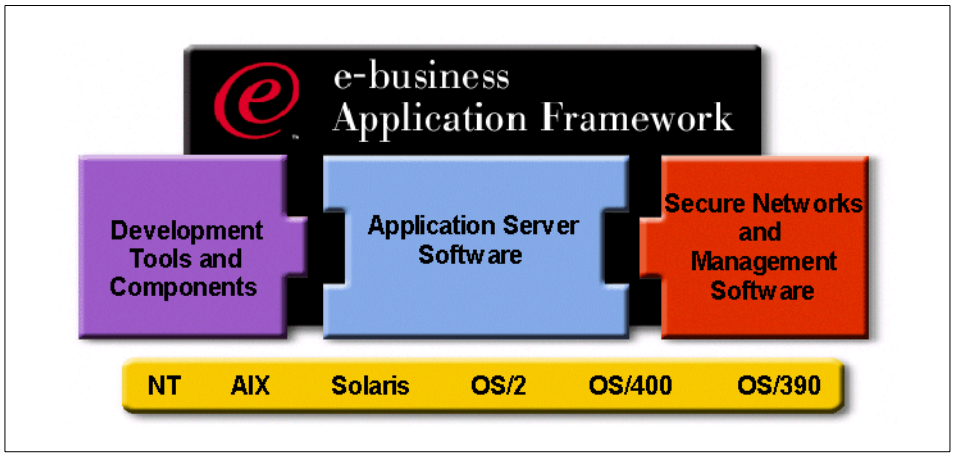


Figure 4. Application framework for e-business

Now that directory standards are emerging, the opportunity exists for enterprises to simplify their directory issues. If an enterprise is able to store directory information once, the total cost of maintaining this information can be significantly reduced. However, in some enterprises, specific application requirements and migration costs may dictate that multiple directories continue to be maintained. In these environments, directory standards enable central administration and reduce the cost and complexity of synchronization.

Given the distributed and heterogeneous nature of network computing, the Application Framework for e-business directory architecture is defined to satisfy the following requirements:

- Directory services must be based on existing or emerging standards when available.
- Access to directory services must be available from a variety of client platforms through a standardized API and standard Web browsers.
- A common schema for universal objects, such as users and groups, must be standardized to eliminate application dependence on specific directory implementations.
- Information maintained in existing directories must be accessible without requiring a complete migration to a new directory.
- Central administration of common objects stored in multiple different directories must be provided.
- Directory services must be secure. Access to information in the directory must be guaranteed for authorized users and denied for non-authorized

users. In addition, the directory architecture must allow for storage and management of security information.

- Directory services must be available when needed by applications.

As part of the Application Framework for e-business, IBM has defined a standards-based directory architecture to address the requirements of directory-based enterprise computing. The architecture contains the following key elements:

- Standards-based LDAP client and server

  LDAP, the lightweight directory access protocol, was developed to provide standards for accessing the data in network directories. The LDAP distributed architecture supports scalable directory services with server replication capabilities (being defined for LDAP Version 3) ensuring that directory data is available when needed. For security, LDAP supports both basic client authentication using a distinguished name and password and SSL (Secure Socket Layer), which provides mutual authentication between clients and servers as well as data security via encryption. LDAP Version 3 supports the Simple Authentication and Security Layer (SASL), a framework for adding additional authentication mechanisms. (More information on security can be found in Chapter 5, “Directory Security” on page 103.)

- Common directory schema

  A directory schema defines the rules by which objects are stored and retrieved in the directory. A common schema allows different applications to share the same set of objects (user, printer, etc.) such that the information for a person or resource is not created and stored in multiple places in different formats within the network. In addition, a common schema definition allows directory-enabled applications to be developed independently of a specific directory implementation. More detailed information about schemas can found in Chapter 2, “Schema and Namespace” on page 31.

- Meta directory

  To ensure interoperability with non-LDAP based directory and synchronization of their contents, a set of meta directory functions is defined. The meta directory serves two purposes: it is used as the *master* for directory information by all other directories and it synchronizes the information between the different directories in an organization allowing users accessing any of the directories to see the same information. The meta directory provides the basis for common administration of the

information stored in the directory by different applications, such as configuration and security data.

1.6 IBM Directory Strategy

To address the proliferation of application-specific directory services, it is IBM's strategy to consolidate its directory offerings onto a standards based LDAP directory service. In addition, IBM will directory-enable a wide range of products to reduce the administrative effort required to maintain them and to improve the level of data consistency across IBM product offerings.

IBM's directory strategy comprises four principles:

1. Directory enable IBM and ISV products.

   Several IBM product groups and selected ISVs (Independent Software Vendors) work together to exploit the IBM SecureWay Directory since the benefits for customers proportionally grow as a common directory is exploited across operating systems, networks, and applications. IBM offerings will be able to use this LDAP directory to store user, configuration, and security information reducing administrative costs and improving end users access to information.

   Though this is not a formal announcement, the following examples show how this can be achieved. IBM networking products can store configuration information in the directory so that each individual device can load its configuration from a central directory service. IBM firewall products can store policy information in the directory so that the policy can be quickly updated throughout the enterprise. IBM middleware products can exploit the directory to maintain a common base of users and security information that can be shared across applications. Once the various levels of the product stack are directory enabled, the synergy between components can be leveraged to deliver powerful benefits. A user running a transaction can be given a high priority on the transaction server and priority on the network to provide improved end-to-end response time. This can be made possible by providing a common directory service that can be leveraged by multiple components. As more and more products become directory-enabled and share the same data, the possibilities for providing cross-product synergy become limitless.

2. Provide a highly scalable cross platform LDAP directory

   A common LDAP server will be included as a core part of IBM operating systems and software suites; so, customers can be certain that an LDAP directory is available. IBM is developing the LDAP- and DB2-based IBM SecureWay Directory to provide a very scalable, high performance

directory service for the wide range of directory-enabled products IBM will be delivering as well as providing leading directory support for third party directory-enabled solutions. The choice of a relational database as a backing store will provide high levels of scalability and reliability. For customers that use a Lotus Domino/Notes environment in their organizations, their Lotus Domino directory conforms to the LDAP standards beginning with Domino Release 5.

IBM also offers the eNetwork X.500 Directory for customers that need an LDAP directory that is also compliant with the X.500 standard. This directory is available on the AIX platform.

3. LDAP support across existing IBM directories

   IBM supports LDAP across the existing directory servers to provide consistent access for application developers and clients. This will provide customers (ISVs, corporate developers, and users) with a single API and protocol for IBM and Lotus-based networks. IBM is also further improving the operational characteristics of LDAP support by providing a common schema across the IBM, Lotus, and Tivoli directories, which is expected to be consistent with the networking-oriented schema that IBM is actively working on as part of the Desktop Management Task Force (DMTF) Directory-Enabled Networks (DEN) initiative.

4. Provide directory management tools

   With the IBM SecureWay Directory (server and client SDK), IBM ships the graphical Directory Management Tool (DMT) that allows an administrator to easily browse the directory tree and perform common administration tasks (see 7.1, “The Directory Management Tool” on page 159).

   In environments where Tivoli is deployed, support for a heterogeneous directory environment through the Tivoli User Administration is provided. Today, this component allows Tivoli to manage NT, NetWare, Domino, various UNIX directories, and the OS/390 Security Server. Through the LDAP Connection (see 1.7.4, “Tivoli User Administration: LDAP Connection” on page 26), the Tivoli User Administration is even capable of managing information stored in an LDAP directory through the versatile administration user interface.

1.7 IBM Directory Offerings

As explained in the previous section, IBM product development groups work on developing directories as well as directory enabling other products to exploit LDAP. Both the IBM SecureWay Directory and Lotus Domino are LDAP Version 3-conforming directory servers and are based on the

requirements in each particular environment that the selection for the directory infrastructure can be made. In some situations, the infrastructure can contain multiple directory servers that might even be using a mixture of IBM SecureWay Directory and Domino servers.

1.7.1 The IBM SecureWay Directory

The IBM SecureWay Directory implements the IETF LDAP V3 specifications. It also includes enhancements added by IBM in functional and performance areas. This version uses the IBM DB2 as the backing store to provide per-LDAP operation transaction integrity, high performance operations, and on-line backup and restore capability. It interoperates with the IETF LDAP V2 based clients. Other major new features include:

- A graphical directory management tool – The IBM SecureWay Directory Management Tool (DMT) allows an administrator to easily browse the directory and add or edit objects, such as object classes, attributes, and entries.
- A dynamically extensible directory schema – This means that administrators can define new attributes and object classes to enhance the directory schema. Changes can be made to the directory schema, too, which are subject to consistency checks. Users may dynamically modify the schema content without rebooting the directory server. Since the schema itself is part of the directory, schema update operations are done through standard LDAP APIs. The major functions provided by LDAPv3 dynamic extensible schema are:
  - Queryable schema information through LDAP APIs
  - Dynamic schema changes through LDAP APIs
  - Server rootDSE
  - Dynamic configuration changes using LDAP APIs
- The Directory-Enabled Network (DEN) schema – The DEN specification defines a standard schema for storing persistent state and an information model for describing the relationships among objects representing users, applications, network elements and networking services. More about schema and namespace is described in Chapter 2, “Schema and Namespace” on page 31.
- Subclassing – Supports object inheritance for object class definitions and attribute definitions. A new object class can be defined using parent classes (multiple inheritance) and the addition or change of attributes. Schema update operations are checked against the schema class hierarchy for consistency before being processed and committed.

- UTF-8 (Universal Character Set Transformation Format) – An IBM SecureWay Directory server supports data in multiple languages and allows users to store, retrieve and manage information in a native language code page. More information about UTF-8 support can be found in 4.3, “UTF-8 Support” on page 83.
- Simple Authentication and Security Layer (SASL) – This support provides for additional authentication mechanisms. SSL provides encryption of data and authentication using X.509v3 public-key certificates. A server may be configured to run with or without SSL support. Security is further detailed in Chapter 5, “Directory Security” on page 103.
- Replication – Replication is supported, which makes additional read-only copies of the directory available thus improving the performance and reliability of the directory service. Replication is further described in 7.6, “Replication” on page 178.
- Referrals – Support for LDAP referrals allows directories to be distributed across multiple LDAP servers where a single server may only contain a subset of the whole directory data. More about referrals is explained in 7.7, “Referrals” on page 184.
- Access Control Model – A powerful, easy-to-manage access control model is supported through ACLs (Access Control Lists, further explained in 5.6, “Access Control” on page 120).
- Support for DNS – An LDAP server can be located through DNS (Domain Name Services) by publishing the Directory server address via Service Resource record in DNS, and clients can find the LDAP server knowing only a symbolic name. See more details in 7.11, “Locating LDAP Servers Using DNS” on page 218.
- Support for server plug-ins – An extensible server architecture which allows implementers to write server plug-ins that carry out additional functions for the server to perform. A plug-in is a dynamic link library (or shared object) that can be included in the server’s address space dynamically. They follow the plug-in APIs published by Netscape, and directory plug-ins are compatible with the Netscape LDAP directory server. Server plug-ins are not further discussed in this book.

The LDAP V3 RFCs implemented in the IBM SecureWay Directory are RFC 2251, RFC 2252, RFC 2253, RFC 2254, RFC 2255, and RFC 2256. You can find more information about these RFCs in Appendix A, “Standards” on page 269.

Administration and configuration for the IBM SecureWay Directory is provided by a Web browser-based GUI. The administration and configuration functions allow the administrator to:

- Perform the initial setup of the directory
- Change configuration parameters and options
- Manage the daily operations of the directory

The IBM SecureWay Directory is available on IBM AIX, Microsoft Windows NT/Intel, and Sun Solaris platforms, and it supports ten languages including English, French, German, Japanese, Simplified Chinese, Traditional Chinese, Korean, Italian, Spanish and Brazilian Portuguese. Catalan is also supported on AIX.

The IBM SecureWay Directory is also available for OS/390 and OS/400. They are integrated in the OS/390 Security Server software and OS/400 Operating System, respectively. The OS/390 and OS/400 implementations are currently on LDAP V2 level.

Platform-specific considerations are described in more detail in Chapter 6, “Installation and Configuration” on page 131.

1.7.2 The IBM SecureWay Directory Client SDK

In addition to the IBM SecureWay Directory, which incorporates an LDAP server, IBM also provides a client development kit that provides the necessary APIs and libraries for development of directory-enabled applications. The IBM SecureWay Directory Client SDK also contains the directory management utilities as explained in Chapter 7, “LDAP Data and System Administration” on page 159.

The IBM SecureWay Directory Client SDK is available for IBM AIX, Microsoft Windows NT/95/98, Sun Solaris. Please check the IBM SecureWay Directory Web site at `http://www.ibm.com/software/enetwork/directory` for the latest information about products, supported platforms, and availability.

1.7.3 Lotus Domino R5.0

Lotus Domino supports a variety of business applications including an LDAP-based directory service. The Domino directory is designed to serve as a key enabling technology for a directory-enabled infrastructure. To leverage the inherent value of this potentially rich and useful store of corporate information beyond e-mail addresses and certificates, Domino/Notes customers can exploit this directory architecture, which they already have in place.

Today's Domino Directory can be part of a general purpose directory infrastructure for the enterprise and for multi-enterprise extranets. On the other hand, in a heterogeneous networking environment with other directory systems, the Domino Directory can also serve as the integration point for directory synchronization, administration, and authentication.

Directory features in Domino R5 include:

- Support for X.500 naming conventions, including hierarchical naming and extensible attributes, for maximum flexibility in configuring the namespace.
- LDAP protocol support in both the client and the server providing lookup (read), add, delete, and modify (write) support for non-Notes clients (for example Web browsers) and servers (for example NDS and Four11).
- Rule-based domain relationships for faster lookups across large namespaces.
- Hierarchical naming and trust between domains to support the relationship of entries across domains.
- Support for a Public Key Infrastructure.
- A dynamically extensible directory schema ideal for customizing the directory to meet specific business requirements.
- Multi-master replication, a key element for reliable directory synchronization and maximum availability.
- An open architecture that can easily incorporate support for emerging standards.

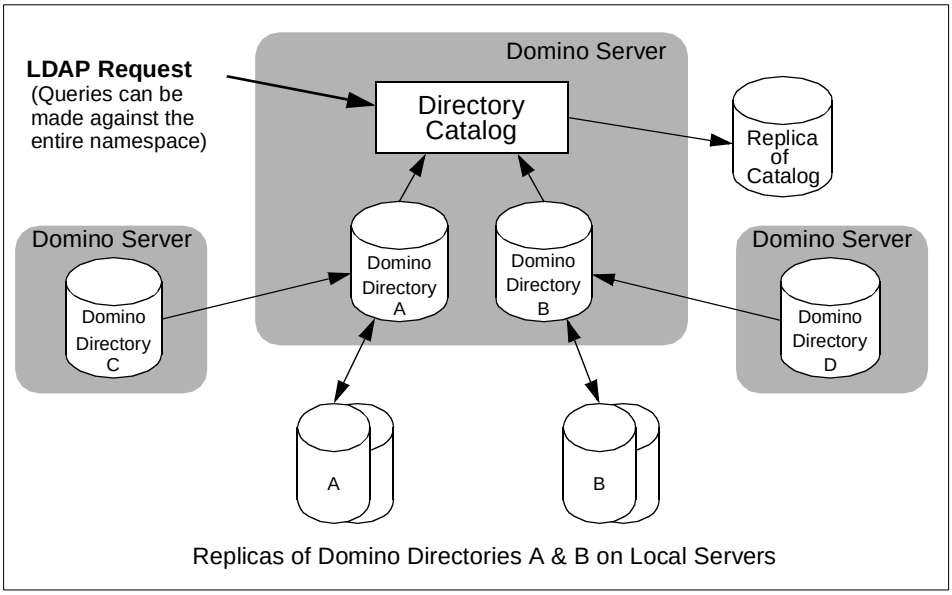


Figure 5. The Domino 5.0 directory architecture

Lotus Domino R5 is available for all the IBM platforms (AIX, OS/2, OS/390, and OS/400), Windows NT, Solaris and HP-UX platforms.

1.7.4 Tivoli User Administration: LDAP Connection

The Tivoli Management Architecture is a highly sophisticated, universal systems management framework available for all major platforms, including Windows NT/95, many UNIX brands and all the IBM platforms.

The Tivoli User Administration (TUA) is based on the Tivoli Configuration and Change Management (CCMS) architecture. The central database, together with a graphical management user interface, allows an administrator to manage user accounts, called profiles, independently of any underlying system. The user information database supports a large number of attributes suitable for almost any kind of target environment. Sets of profiles are maintained within a profile manager. The profile managers distribute profile information to subscribed profile endpoints. The Tivoli LDAP Connection is such a profile endpoint. For example, data in user profiles can travel to LDAP directories via distribute. The reverse process, called population, collects the data from an LDAP directory and stores it in Tivoli user profiles for subsequent management. Figure 6 on page 27 depicts this relationship.

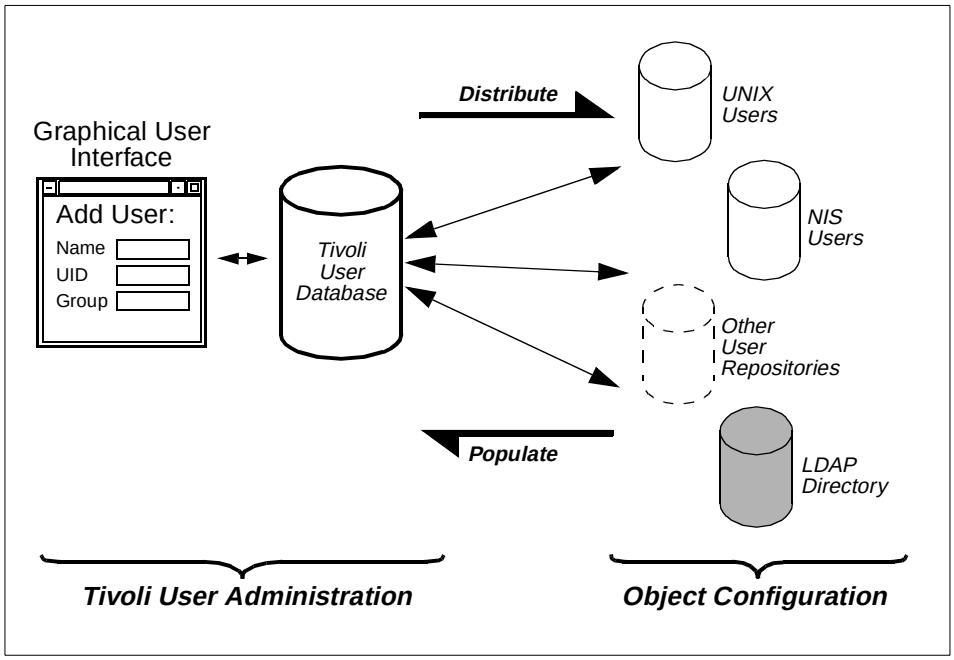


Figure 6. Tivoli database versus the real configuration

Through the addition of an LDAP endpoint, Tivoli User Administration is enriched in its functionality to manage LDAP directory data (while other system-related management and monitoring can be done using other Tivoli systems management and monitoring functionality). This connection runs on any managed node that may or may not be on the same system as the Tivoli Administration Server, which is also called the TMR Server (Tivoli Management Region) Server. Since LDAP is a system-independent standard, the actual LDAP service can be run on any platform. This approach centers the Tivoli User Administration database, and all other user data repositories, including the LDAP directory, are kept in sync with the master Tivoli database. An administrator only uses one common user interface and one single tool to manage user accounts whether the actual users exist on a UNIX system, in an NIS domain, in LDAP, or in any other of the many supported endpoints (Figure 6 above).

It should be mentioned that the Tivoli User Administration also supports a command line interface for all operations. This allows automation of administration tasks from within command language programs.

Tivoli User Administration is supported on AIX V4, OS/390, Windows NT, Solaris 2.5, and HPUX 10.

At the time of writing, the Tivoli LDAP Connection was under the Technology Preview program. The product was supposed to be generally available shortly thereafter.

1.8 LDAP Standards

Several standards in the form of IETF RFCs exist for LDAP. The following is a brief list of RFCs that apply for LDAP Version 2 and Version 3:

LDAP Version 2:

- RFC 1777: Defines the LDAP protocol
- RFC 1778: The String Representation of Standard Attribute Syntax
- RFC 1779: A String Representation of Distinguished Names
- RFC 1959: An LDAP URL Format
- RFC 1960: A String Representation of LDAP Search Filters

LDAP Version 3:

- RFC 2251: LDAP protocol - V3
- RFC 2252: Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions
- RFC 2253: Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names
- 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

More information and a brief description about these standards (RFCs) can be found in Appendix A, “Standards” on page 269.

1.9 Summary

The combination of the IBM cross-platform applications and services integrated on an open standards-based LDAP directory provides an excellent base for directory exploitation projects. As the product lines of IBM and its partners becomes directory enabled, IBM will offer services from the networking layer through the middleware to the end user applications that can share a common base of user, security, and object information based on open standard directory services. This will provide customers with the flexibility to

deploy applications and services across the platforms of their choice and leverage the investment in their current install base of hardware and software. All this will be backed by the experience and reach of the IBM Global Services organization, worldwide services and support infrastructure and IBM experience in providing mission-critical services.

Chapter 2. Schema and Namespace

This chapter includes a comprehensive definition of the *Directory Schema* and *Directory Namespace* for the IBM SecureWay Directory. Some introductory information on the naming and information model is also included along with general guidelines of reuse for objects and attributes. Some examples will be introduced in a simple way, that is, from an exploiter's perspective rather than from a directory architecture perspective in order to help the reader understand how the schema is used.

Tip

The IBM SecureWay Directory Management Tool (DMT) is a graphical tool that is part of the product. It lets you browse and edit various kinds of information in your directory, such as schema definitions, the directory tree, and data entries.

If not already done, we recommend that you install the IBM SecureWay Directory on a workstation and use the DMT to *walk* through the various kinds of objects. It gives you a broad picture and helps you understand the terms used and the relationships between these objects.

2.1 LDAP Information Model Overview

The LDAP information model is based on a subset of the X.500 information model but is extensible and modifiable. Complex attribute types supported in X.500 are not supported in LDAP.

Information is stored into *entries* which contain *attributes*. The allowed set of characters for object and attribute names is defined in *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*, RFC 2252 (for example, the underscore character is invalid). Attributes are typed in the form of <type>=<value> pairs in which the *type* is defined by an *object identifier (OID)* and the *value* has a defined syntax. Attributes can be *single-valued* or *multi-valued*. There is no ordering for multiple values or a multi-valued attribute. An entry can contain multiple attributes. Entries are organized hierarchically by their *distinguished name* (*dn*). Entries whose distinguished name contains the distinguished name of another entry as a suffix are considered to reside *under* the latter entry in the hierarchy, that is, the namespace is *hierarchical*.

Schema defines rules for distinguished names and what attributes an entry must and/or may contain. To organize the information stored in LDAP directory entries, the schema defines *object classes*. An object class consists of a set of *mandatory* and *optional* attributes. Every entry in the LDAP directory has an object class associated with it. Thus, every entry in the LDAP directory contains a set of mandatory attributes and (possibly) a subset of the optional attributes based on the entry's object class and that object class' defined schema. The object class definition also defines where (with respect to other objects in the namespace) it may appear.

An object class is declared as *abstract, structural*, or *auxiliary*. An abstract object class is used as a template for creating other object classes. A directory entry cannot be instantiated from an abstract object class. Directory entries are instantiated from structural object classes. An auxiliary object class cannot be instantiated by itself as a directory entry; it can be attached to directory entries that are instantiated from structural object classes. Auxiliary object classes provide a method for extending structural object classes without having to change the schema definition of a structural class.

Let's look at the following example: Top is an abstract class that contains the objectClass attribute. Person is a structural class that instantiates the directory entry for a given person where the objectClass attribute is also part of that Person entry. So far, this example has used only attributes and object classes defined in a standard. So, now, you may want to tailor the people entries to include information that your company requires and that is not defined in the standard Person object definition. There are two ways to do this:

1. Subclass the Person object to create a new structural class that includes those additional attributes defined by your company, and instantiate the Person directory entry based on that new class.
2. Define that collection of company attributes needed for your company's Person definition as an auxiliary class, and attach it to the directory entry instantiated from the Person class.

Either method is recommended. The downside to auxiliary classes is that if the auxiliary class includes an attribute that is also included in the structural class definition, when that attribute is included in the instantiated directory entry and that attribute contains multiple values and you want to delete the attribute, you cannot tell whether the attribute (when added to the entry) was added when the structural class was instantiated or when the auxiliary class was instantiated. This may be important to the implementor or administrator.

Special entries exist in the namespace called *aliases.* Aliases represent links to other entries or *partitions* of the namespace. When the distinguished name of an alias is used, the entry accessed is the entry to which the alias refers (unless specified otherwise through the programming interface).

The collection of directory entries forms the *Directory Information Tree* (DIT). The method of storage for the DIT of the LDAP directory is implementation-dependent and hidden from the user of that LDAP directory. For example, the IBM SecureWay Directory uses DB2 as its data store, but no DB2 constructs are externalized to LDAP.

2.2 LDAP Names

Entries in the IBM SecureWay Directory are identified by their *names*. The characteristics of these names include:

- They have two forms: a string representation and a URL.
- They have a uniform syntax.
- Namespace boundaries are not apparent in them.

A component of a name is called a *relative distinguished name* (RDN). An RDN represents a point within the namespace hierarchy. RDNs are separated by and concatenated using the comma (“,”). Each RDN is typed. RDNs have the form <type>=<value> for single valued RDNs. The plus sign (“+”) is used to form multi-valued RDNs: <type>=<value>+<type>=<value>.

The <type> is case-insensitive and the <value> is defined to have a particular syntax. The order of RDNs in an LDAP name is the most specific RDN first followed by the less specific RDNs moving up the DIT hierarchy. A concatenated series of RDNs equates to a distinguished name. The DN is used to represent an object and the path to the object in the hierarchical namespace. A URL format for LDAP has been defined that includes a DN as a component of the URL. These forms are explained in the sections that follow.

2.2.1 String Form

The exact syntax for names is defined in RFC 2253. Rather than duplicating the RFC text, the following are examples of valid distinguished names written in string form:

- cn=Joe Q. Public, ou=Austin, o=IBM

  This is a name containing three relative distinguished names (RDNs).

- ou=deptUVZS + cn=Joe Q. Public, ou=Austin, o=IBM

This a name containing three RDNs in which the first RDN is multi-valued.

- `cn=L. Eagle, o=Sue\, Grabbit and Runn, c=GB`

  This example shows the method of quoting a comma (using a backslash as the escape character) in an organization name.

- `cn=Before\0DAfter,o=Test,c=GB`

  This is an example name in which a value contains a carriage return character (0DH).

- `sn=Lu\C4\8Di\C4\87`

  This last example represents an RDN surname value consisting of five letters (including non-standard ASCII characters) that is written in printable ASCII characters. Table 2 explains the quoted character codes.

Table 2. The ASCII encoding of an RDN surname (example)

Unicode Letter Description	ISO 10646 Code	UTF-8	Quoted
Latin capital letter *L*	U0000004C	0x4C	L
Latin small letter *u*	U00000075	0x75	u
Latin small letter *c* with caron	U0000010D	0xC48D	\C4\8D
Latin small letter *i*	U00000069	0x69	i
Latin small letter *c* with acute	U00000107	0xC487	\C4\87

For the detailed definition of DNs in string form, consult RFC 2253. More about Unicode character encoding (superset of ISO 10646) and its transformation into UTF-8 can be found at `http://www.unicode.org` and in RFC 2279.

2.2.2 URL Form

The LDAP URL format has the general form ldap://<host>:<port>/<path>, where <path> has the form <dn>[?<attributes>[?<scope>?<filter>]].

The <dn> is an LDAP distinguished name using the string representation (see previous section). The <attributes> indicates which attributes should be returned from the entry or entries. If omitted, all attributes are returned. The <scope> specifies the scope of the search to be performed. Scopes may be current entry, one-level (current entry's children), or the whole subtree. The <filter> specifies the search filter to apply to entries within the specified scope during the search. The URL format allows Internet clients, for example, Web browsers, to have direct access to the LDAP protocol and thus LDAP directories.

Examples of LDAP URLs are:

- ldap://austin.ibm.com/ou=Austin,o=IBM

  This URL corresponds to a base object search of the <ou=Austin, o=IBM> entry using a filter <of objectClass=*> requesting all attributes (if a filter is omitted, a filter of <objectClass=*> is assumed by definition).

- ldap://austin.ibm.com/o=IBM?postalAddress

  This is an LDAP URL referring to only the postalAddress attribute of the IBM entry.

- ldap:///ou=Austin,o=IBM??sub?(cn=Joe Q. Public)

  This is an LDAP URL referring to the set of entries found by querying any capable LDAP server (no hostname was given) and doing a subtree search of the IBM Austin subtree for any entry with a common name of Joe Q. Public retrieving all attributes.

The LDAP URL format is defined in RFC 2255.

Using URLs for Directory Updates

Although the LDAP URL format basically allows updates (not only reads and searches) to be performed against an LDAP directory, its use for updates is discouraged until adequate security methods are mandated and/or implemented.

2.2.3 Additional Syntaxes

The IBM SecureWay Directory supports the syntax described above. In the future, additional syntaxes for distinguished names may be supported. Using the simple LDAP syntax example <cn=Joe Q. Public, ou=Austin, o=IBM>, the additional name syntaxes can include:

- DCE (OSF) form

  RDNs are ordered from least significant to most significant and are slash (/) separated: /o=IBM/ou=Austin/cn=Joe Q. Public

- Reverse LDAP form

  RDNs are ordered from least significant to most significant, but are still comma (,) separated: o=IBM, ou=Austin, cn=Joe Q. Public

2.3 Directory Information Tree (Namespace) Structure

This section explains the basics for the structure of the namespace. The key for understanding the relationship between the namespace and the schema is that there is no pre-defined rigid artificial namespace, but, rather, the power of the object class definitions is used (for example the objectClass attribute, DIT structure hierarchy, and so on) to allow the exploiter to define an explicit shape of the namespace. Neither are there any artificial intermediate nodes in the namespace hierarchy to segregate subtrees within a partition of the namespace. However, *container nodes* are used where appropriate either because the directory administrator chooses to exploit them or because they provide useful isolation within an application/product-constructed portion of the DIT. A container is essentially an empty node in the namespace that can be used to partition the namespace. Analogous to a file system directory, a container can have directory entries beneath it in the DIT structure.

So, continuing with the example, <cn=John Smith, ou=Austin, o=IBM> is a natural form for John Smith. The objectClass attribute (which is part of every directory entry) is used to specify the type of entry. The data store, which holds the directory entries and their attributes, is trusted to organize the storage of the data in a performance-efficient manner transparent to the user of the directory data.

This results in a *hierarchical structure* where the names of directory entries can be natural and meaningful. The directory administrator or designer is provided flexibility in structuring the data in the directory.

To understand the namespace structure, a partition of the global namespace is divided into three logical pieces:

- The *suffix*, which defines (names) the root of a partition of the global namespace. Other directory servers are interested in the suffix because it states the partition that this directory server holds. In other words, the suffix is the top-most entry of a partition that a server stores. A server can serve more than one partition of a namespace, in which case, there is more than one suffix.
- The organizational structure(s) of the namespace (a *hierarchy*) underneath that suffix. The hierarchy can be flat (one single level underneath the suffix) or branched to several levels.
- The *directory data* stored in some hierarchical manner (either flat or hierarchical) within the two logical pieces (suffix and organizational structure) of the partition.

Figure 7 on page 37 depicts the relationship between the conceptual namespace organization and a practical implementation.

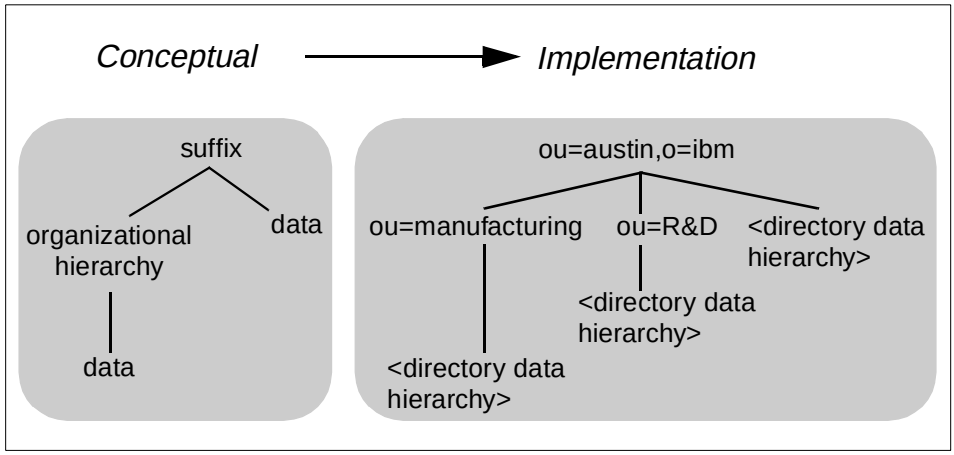


Figure 7. Namespace organization

The suffix for a specific directory server can be defined to be a DN that spans any part of those three logical pieces and the directory server's data beneath that suffix. On the other hand, if the most-significant RDN of the suffix's DN is a leaf, there is no data held by that directory server's partition.

In the IBM SecureWay Directory, the schema definition is part of the directory itself in the subschema object class which is discussed in more detail in 2.6.1, "Directory Server Objects" on page 45.

2.4 Relationship Between Objects

The associations between two directory entries can be defined in two ways:

- By *containment*: the DIT hierarchy
- By *reference*: the use of attributes of type (syntax) distinguished name (DN)

The directory entry associations provide both logical correlation (for example, an operating system is part of a computer system and a user is authorized to use certain system resources) and directory traversal paths for the operations supported by the schema. The use of the DIT containment hierarchy permits certain searches to be made more efficiently; searches may be scoped and filtered to retrieve all of a related set of objects in one LDAP request.

Traversals based on DN pointer associations, on the other hand, cannot be grouped this way, but the associations may be many-to-many.

The simplest association between objects is established by their structure in the directory information tree. This is analogous to the structure of a file system directory tree. The DIT hierarchy is a natural method for aggregate associations (for example: *part of*), but it can also be used for other associations that are one-to-many. As noted above, associations encoded in the DIT can provide retrieval of a collection of entries in a single LDAP call. This use must be balanced against the requirement for flexibility in the directory administrators' choices for DIT structure.

A distinguished name can be thought of as a directory pointer, and, as such, can provide associations between directory entries. The value of an attribute that is defined as syntax DN can be said to point to an entry named by that DN. The entry that the DN pointer names may be in the same partition or in a different partition as the directory entry that holds the DN pointer. For example, an account entry may contain the DN of the white pages (person) object for the user's account entry. This is called a *forward pointer*, and it represents an association from the account object to the person object. Additionally, the object pointed to (by the DN pointer) may also contain a pointer back to the originating object. Continuing with the preceding example, the white pages person object may contain a DN pointer back to the account object. This is called a *backward pointer* (or simply *back pointer*). A back pointer in our example represents an association from the person object back to the account object.

Both forward and backward pointers are considered valid if there is a method by which referential integrity can be maintained (a guarantee that the values of each pointer are always valid such as via a transaction). However, this complicates the application since referential integrity is not supported by the directory server itself and must be handled by the exploiting application. If a back pointer is defined, the integrity of its values is not guaranteed. In this case, the back pointer can only be used as a hint or assertion. These back pointers may be used to optimize traversals through the directory, and they may be used to assert a relationship between objects, but it is the forward pointer that establishes the valid relationship between the objects. It is expected that the application must be written to handle the failure case, for example, to reissue an operation that will then search or use a forward pointer to obtain the information for which it was searching. Optionally, the application may repair a back pointer it finds invalid. (The exploiting applications should, however, not run background daemons that constantly look for and repair any invalid pointers. Utilities of this nature usually do not scale very well and can be CPU-intensive.) Where practical, use forward

pointers and the DIT structure to establish associations between objects and to avoid the use of back pointers. When back pointers are used, they should be clearly identified as such (for example, as back pointer, hint, or assertion).

2.4.1 Object and Attribute Reuse

There are two kinds of reuse in developing a directory schema: reuse *by subclassing* from existing directory object classes and reuse *by applying* previously defined attributes in additional (or new subclass) object classes. Reuse of object class and attribute definitions helps to eliminate redundancy and, hence, helps to produce a more understandable and usable schema. Such a schema encourages exploitation of the directory and reuse.

The base schema that IBM ships with the IBM SecureWay Directory already defines many object classes and attributes. In developing any extensions to the common schema, use the definitions already in the schema where possible. These include the industry standard definitions that IBM supports plus definitions that IBM has developed for additional use. But, you may discover that you will probably need to define some new objects and attributes to put your information into the directory service. You should then subclass the objects where possible and define new objects only when the current ones do not meet your needs. Likewise, examine the attributes in the existing schema. Just because an attribute is used in an object class that is of no interest to you should not hinder you from still using (reusing) that attribute in your object class. This is because the object class is just a mechanism for defining a collection of attributes for the instantiation of a directory entry.

Let's look at an example. Suppose there is an object class dog that includes the attributes objectClass, cn, breed, description, weight, and gender. These attributes are known within the directory server so anyone can use (re-use) these attributes in other object classes within that same directory server. Next, define a new object class human. You define it to have attributes objectClass, cn, weight, gender, height, and race. Note that you have re-used the attributes objectClass, cn, weight, and gender and defined only the new attributes; height and race. The objectClass attribute is reused because all objects are subclasses of top; the cn, weight and gender attributes can be reused by subclassing if there is a common superclass for dog and human (perhaps mammal) or, if there is no common superclass with cn, weigh and gender, by simply adding the already defined attributes to the newly defined human object class.

Object classes and attributes are not the only elements that can be reused. The schema also defines matching rules and syntaxes. And, in fact, you must

use (reuse) those that are defined. It is also possible to define your own matching rule or syntax.

2.4.2 Naming Conventions

Section 2.2, “LDAP Names” on page 33 introduced the formal LDAP grammar for naming entries in the IBM SecureWay Directory. In addition to these rules for syntaxes and forms, there are several guidelines or conventions for naming entries in the IBM SecureWay Directory. Some of these conventions, such as the desire to use object names that connote meaning to a human reader, are driven more by common sense than any real programming considerations. On the other hand, a few of the naming conventions described below are needed precisely for programming or operational reasons. Naming conventions exist for LDAP names that have the following goals:

- A user (directory designer, exploiter, or application programmer) can understand the conventions used to name the objects and attributes in the IBM SecureWay Directory.
- A user can use the same conventions for naming new schema for his/her own applications as applicable.

The naming conventions that were used to define the IBM schema are:

1. Distinguished names (DNs) should be relatively short (fewer number of components).

   This is achieved by defining the DIT to be as flat as possible. The <cn=John Smith> example used in the above sections is an example of using a short(er) DN by making use of the objectClass attribute value.

2. Object and Attribute names should be user friendly.

   For example, consider that the attribute name for a fax phone number is facsimileTelephoneNumber rather than an abbreviation, such as faxnum.

3. Distinguished names should connote meaning.

   Consider the following two examples:

   - cn=4019ps, cn=lpt1, cn=lan5, ou=Austin, o=IBM
   - printerModel=4019ps, portNumber=lpt1, networkName=lan5, ou=Austin, o=IBM

   The second example using descriptive attribute names as part of the RDN connotes more meaning than the first, although both may be grammatically correct.

4. Take advantage of multi-valued relative distinguished names (RDNs).

One of the recommendations contained in RFC 1617, *Naming and Structuring Guidelines for X.500 Directory Pilots* (by P. Barker, S. Kille, May 1994) is that multi-component (also called multi-valued) RDNs is a useful way to ensure uniqueness of an entry, especially in a flat DIT. <ou=deptUVCS + cn=Joe Q. Public, ou=Austin, o=IBM> is an example of an RDN where the first RDN is multi-valued.

5. Use objects and attributes that are defined by standards organizations.

   Special care must be taken to ensure semantic and syntactic consistency and precision between the standard definition and your implementation. You must ensure that standard names are used when you implement an object or attribute that is likely to become part of an industry solution (that is not just your own solution) so that an industry client application can locate entries in your directory implementation.

   Many object and attribute names are defined by more than one standards body. In many cases the definitions have been kept common across the standards bodies and industry groups involved in defining directory schema. Where inconsistencies arise, choices are made based on market penetration expectations, model consistency, and the defining specifications (for example X.500, IETF, CIM, and so on).

6. For IBM-specific objects and attributes, IBM uses the *eObjectname* or *eAttributename* naming convention.

   The lower case *e* prefix convention is used for human readability only. In naming convention 5 (above), IBM defined the convention that standard names must be used when defining objects that may be used by an industry solution. The reverse is also true. Object and attribute names that have been defined by standards groups should not be used by implementation projects to define *other* or *similar* objects or attributes. IBM chose the *e* prefix as a convention to help ensure that name collisions will not occur with either standard names or with names chosen by other directory server vendors or other directory-enabled products that define schema. The *e* prefix is an IBM convention and does not guarantee that there will not be any collisions with any other vendor's implementation or future definitions within the standards.

2.5 The IBM Schema

With the IBM SecureWay Directory, IBM provides an extensive schema called the *IBM schema*. The IBM schema is comprised of object classes and attributes defined by industry standards as well as object classes and attributes defined by IBM that are either derived from industry standards or new. The IBM schema is not a static schema, and new classes and attributes

will be added as requirements dictate. Figure 8 on page 42 shows the main object classes that IBM has derived from industry schemas that form the basis for the IBM SecureWay Directory. This overall view is quite complex; so, it is also dissected into smaller sections and discussed in the succeeding schema sections.

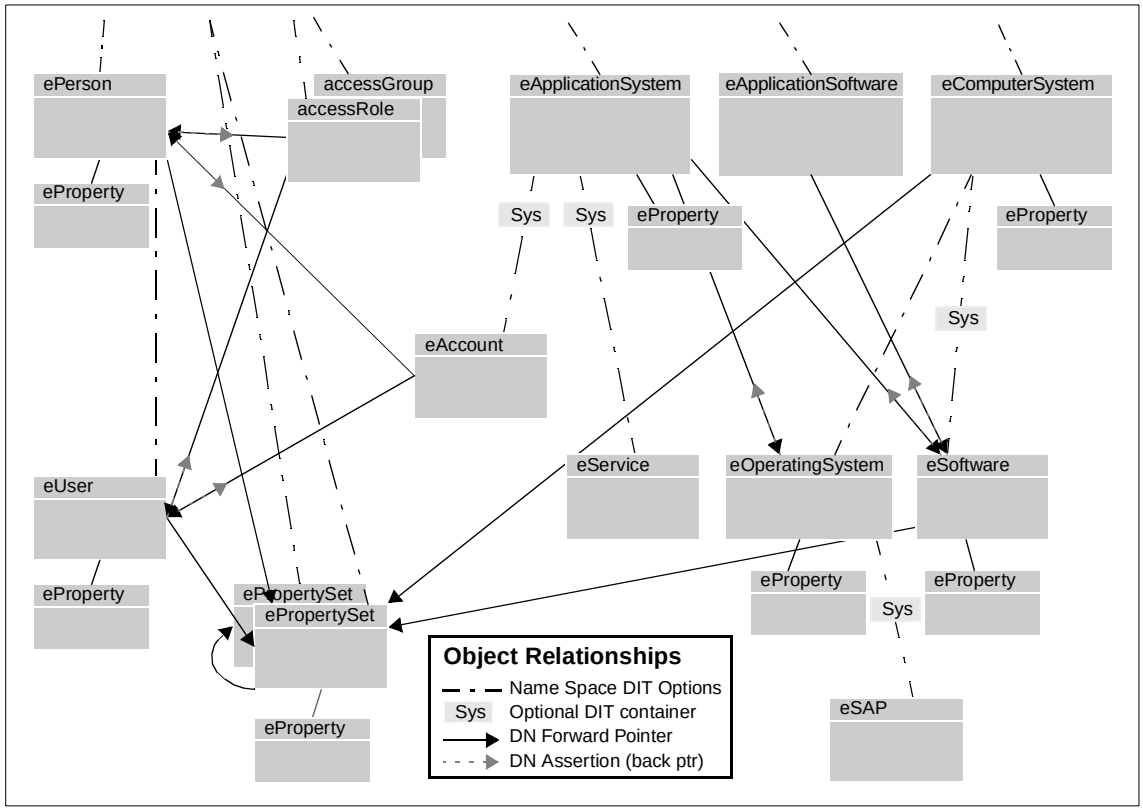


Figure 8. The big picture of the IBM schema

Each of the subtrees may appear in the namespace per the previous section that discussed DIT structure.

The IBM SecureWay Directory schema is based on industry standards where standards exist. It also extends the industry schema as necessary so that IBM products can exploit the directory service and management utilities can manage that data. As the schema for exploiting products is defined, it will be folded in the base schema shipped with the IBM SecureWay Directory. The industry standards on which the IBM schema is based include:

- Desktop Management Task Force's Common Information Model, CIM (`http://www.dmtf.org`).

  CIM defines schema for managed elements in a system. These consist of physical objects such as computer systems, software objects, devices, network elements, user related elements, security elements, policies, and so on. IBM incorporates these definitions and creates subclasses from them when reasonable. The Directory-Enabled Networks (DEN) schema is incorporated by CIM.

- Internet Engineering Task Force, IETF (`http://www.ietf.org`).

  There are several RFCs that define schema for LDAP V2 and LDAP V3: RFC 1778, 2252, 2256. These RFCs are derived from industry experience and X.500 standards. The set IBM supports in its IBM SecureWay Directory are mainly the objects and attributes related to country, organization, person, group, and their subclasses. There are some object classes and attributes that IBM lists in its schema from RFC 1274, although many of them are superseded by the RFCs listed above.

- Network Application Consortium's Lightweight Internet Person Schema, LIPS (`http://www.netapps.org`).

  LIPS is an industry consensus definition of a person. It includes information pertaining to *white pages, organization, residence, phone numbers, IDs and passwords*, and more. This schema has also been adopted and standardized by The Open Group and specified in the Internet White Pages profile (a profile that directory servers may support). The LIPS object class, along with its attributes, is a subclass of the X.500 person object class.

2.5.1 Schema Information

The schema defined is stored in a special directory entry cn=schema. The schema contains the following information:

- *Object class* is a collection of attributes. A class can inherit attributes from one or more parent classes.
- *Attribute types* contains information about the attribute, such as name, OID, and matching rules.
- *IBM attribute types* are the implementation-specific attributes, such as database table name, column name, SQL type, and max length of each attribute.
- *Syntaxes* include:
  - Distinguished Name

- Telephone Number
- Binary
- IA5 String
- Directory String
- Boolean
- Integer
- Generalized Time and UTC Time

- *Matching Rules* supported in the IBM SecureWay Directory V3.1 server are those defined in the LDAPv3 specifications for each syntax.
- The syntax for schema definitions complies with what is defined in RFC 2252.

2.6 Schema Categories

The schema provided by IBM is based on the industry work (see previous section) in addition to what is needed to support IBM products that exploit the directory service. The schema is logically divided into several categories for ease of understanding and discussion:

- *directory server objects* such as *top* and *subschema* (also, some object classes included in this category are for convenience, although they are not just used by a directory server).
- *white pages,* which includes objects such as *person*, *group*, *country*, *organization*, *organizational unit* and *role*, *locality*, *state*, and so on.
- *security,* which includes the objects necessary for authorization, authentication, accounting, and audit.
- *policy* and *profile,* which include objects for security- and non-security-related policies and profiles.
- *system,* which includes objects such as *application*, *computer*, and *operating systems*.
- *software,* which includes objects such as *software inventory*.
- *services,* which includes objects such as *services*, and *access points*.
- *other,* which includes objects that are IBM-defined and not yet categorized.

In the sections that follow, each of these schema categories is briefly explained with a description of its object classes, the (noteworthy) attributes of that object class, and its relationships to other object classes.

2.6.1 Directory Server Objects

The directory server objects are those that the directory server maintains for storing information about itself and other directory servers. It includes the following object classes:

- top
- subschema
- extensibleObject
- replicaObject
- referral
- cacheObject
- container
- linkedContainer

The top object class is an abstract class. All other object classes are subclasses of top. top has just one mandatory attribute: the objectClass attribute. This attribute says what type of object a particular directory entry describes. All entries in the directory must have an objectClass attribute because they are subclasses of top.

The subschema object class is an auxiliary object class. It contains the schema (for example object classes, attribute types, matching rules, and so on) for the LDAP directory server. As an auxiliary object class, it can be attached to any structural object class that is instantiated on a directory entry. This object class defines what schema is allowed where in the DIT. At minimum, this object class is found at the top of a server's DIT, that is, at the suffix directory entry.

The extensibleObject object class is an auxiliary object class. It contains every attribute defined by a directory server's schema. IBM supports this object class for completeness but discourages its use since it basically permits any attribute to be added to any directory entry (which is not conducive to a structure-enforced DIT).

The replicaObject object class is an IBM-defined structural class that is used to represent a directory server replica. It contains attributes used to control directory server replication. This object and its attributes are used only by the directory server and its administration/management utilities; other products do not use this object.

The referral object class is a structural object class that presents a referral directory entry. It contains a single attribute, ref, that specifies a reference to another part of the global namespace in URL format.

The cacheObject object class is an auxiliary object class that allows a time-to-live attribute, ttl, to be associated with a directory entry. This enables a client to know how long it may cache that directory entry's information before it must re-fetch another copy of that directory entry from the directory server.

The container object class is a structural object class that can contain other objects. It is used to provide subtree grouping in the namespace. A container object is analogous to a file system directory. Objects associated with the directory container are hierarchically beneath that container in the namespace (see also 2.3, "Directory Information Tree (Namespace) Structure" on page 36).

The linkedContainer object class is an abstract object class with a DN-valued property pointing to another container to search if the desired object is not found in the current container. This class is a DEN class and is used to build content hierarchies.

2.6.2 White Pages Objects

The white pages schema is analogous to telephone directory information. It is meant to describe people and organizations. It contains the following objects:

- Organizational objects:
  - country
  - locality
  - organization
  - organizationalUnit
  - dcObject
  - domain
  - liOrganization
- Person objects:
  - person
  - organizationalPerson
  - inetOrgPerson
  - residentialPerson
  - liPerson
  - ePerson
- Group objects:
  - groupOfNames
  - groupOfUniqueNames
- Miscellaneous objects:

- alias
- aliasObject
- organizationalRole

All these objects are defined by industry standards, except for the ePerson and aliasObject objects, which are defined by IBM. The following sections describe these objects in more details.

2.6.2.1 Organizational Objects

The organizational objects consist of several structural object classes, each of whose superior is the top object class in the class hierarchy (except the liOrganization class, which is subclassed from organization).

The country object class is used to define country entries in the DIT. The locality object class is used to define locality in the DIT. The organization object class is used to define organization entries in the DIT. The organizationalUnit object class is used to define entries that represent subdivisions of organizations.

The dcObject and domain object classes in *Using Domains in LDAP/X.500 Distinguished Names*, RFC 2247, are used to map domains into distinguished names. The first, dcObject, is intended to be used in entries for which there is an appropriate structural object class. For example, if the domain represents a particular organization, the entry would have organization as its structural object class, and the dcObject class would be an auxiliary class. The second, domain, is a structural object class used for entries in which no other information is being stored. The domain object class is typically used for entries that are plan holders or whose domains do not correspond to real-world entities. The dcObject object class permits the dc attribute to be present in an entry. This object class is defined as auxiliary, because it would typically be used in conjunction with an existing structural object class, such as organization, organizationalUnit or locality. If the entry does not correspond to an organization, organizational unit, or other type of object for which an object class has been defined, the domain object class can be used. The domain object class requires that the DC attribute be present and permits several other attributes to be present in the entry. The DC attribute is used for naming entries of the domain class.

The liOrganization object class is a structural object class that is subclassed from the organization object class. It defines organizations in a manner acceptable for the Internet in a lightweight fashion.

2.6.2.2 Person Objects

The person objects consist of several objects subclassed from the person object. The Figure 9 on page 48 shows the class hierarchy (subclassing).

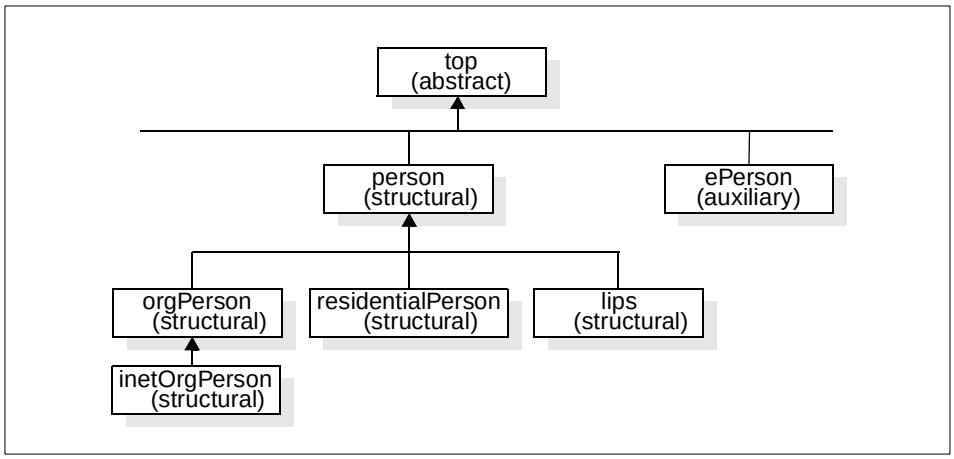


Figure 9. Class hierarchy for person object classes

The person object class is a structural object class defined by X.521 and RFC 2256 (same definition). It defines entries representing people generically. It contains the minimal set of attributes that the directory entry of type object class=person must contain: commonName (or cn) and surname (or sn) plus some additional optional attributes.

The organizationalPerson object class is a structural object class subclassed from person and is also defined by X.521 and RFC 2256 (same definition). It defines entries representing people employed by or associated with an organization. It adds optional attributes to those already defined in the person object class.

The inetOrgPerson object class is a structural object class subclassed from organizationalPerson. It is defined in an Internet Draft (not yet ratified). It defines entries representing person information requirements found in typical Internet and intranet directory service deployments. It incorporates attributes needed to define both organizational and residential characteristics of a person. It adds optional attributes to those already defined in the person and organizationalPerson object classes.

The residentialPerson object class is a structural object class subclassed from person and is also defined by X.521 and RFC 2256 (same definition). It

defines entries representing people in a residential environment. It adds optional attributes to those already defined in the person object class.

The liPerson (lightweight Internet person) object class is a structural object class subclassed from person and is defined by the Network Applications Consortium and The Open Group (same definition). It defines entries representing people and contains the commonly used organizational and residential attributes that have been accepted by a wide group of companies. It adds these optional attributes to those already defined in the person object class.

The ePerson object class is an auxiliary object class defined by IBM (the *e* prefix identifies it as an IBM extension, see 2.4.2, “Naming Conventions” on page 40) and subclassed from top. It may be attached to any person directory entry, that is, a directory entry instantiated from any of the person structural classes: person, organizationalPerson, inetOrgPerson, residentialPerson, or liPerson. The ePerson object class supplements the existing person class directory entries with attributes needed for IBM software. When attached to an instantiated directory entry, that directory entry has the characteristics presented by the attributes of its structural class plus those attributes defined by IBM. If no structural class is specified when a new person needs to be instantiated, the default class used is inetOrgPerson, otherwise the existing entry for that person is used. This allows IBM software to use any customer or vendor chosen person class, that is, IBM software can depend on the ePerson object class independently of what others may use as their person structural class. For example, Microsoft has defined its own person class for Windows NT users as structural. IBM can then append it’s ePerson definition to this NT class, and, thereby, not disturb the NT definition while allowing its software products based on ePerson to continue to work without modification. Queries for object classes of type ePerson will return any directory entry to which the ePerson auxiliary class was appended.

2.6.2.3 Group Objects

The Group object classes are defined by X.521 and RFC 2256 (same definition). Figure 10 shows the class hierarchy (subclassing).

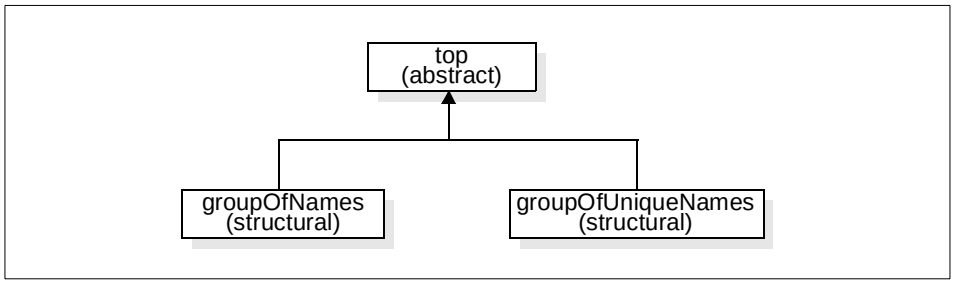


Figure 10. Class hierarchy for group object classes

The groupOfNames object class is a structural object class. It is used to define entries representing an unordered list of names that represent individual objects or other groups of names. The membership of a group is static, that is, it is explicitly modified by administrative action rather than dynamically determined each time the group is referenced. The membership of a group can be reduced to a set of individual object's names by replacing each group with its membership (expansion of nested groups). Examples of groups are e-mail lists or department memberships.

The groupOfUniqueNames object class is a structural object class. It is the same as groupOfNames except that the integrity of its unordered list of names can be assured.

2.6.2.4 Miscellaneous Objects

The miscellaneous object classes are defined by X.521 and RFC 2256 (same definition). Figure 11 below shows the class hierarchy (subclassing).

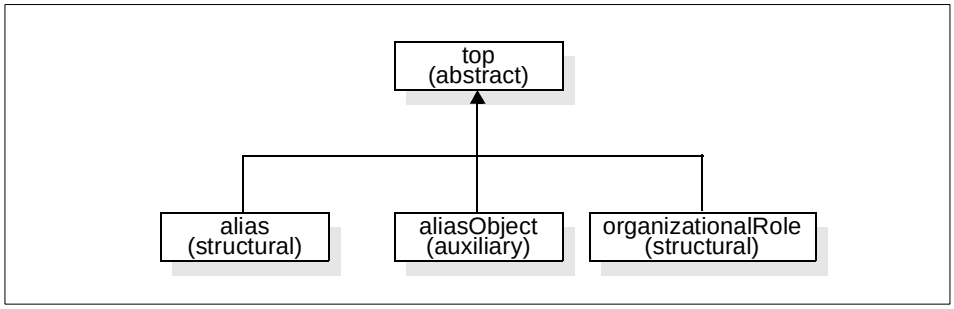


Figure 11. Class hierarchy for miscellaneous object classes

The alias object class is a structural object class whose superior in the class hierarchy is the top object class. It allows an object to be known by more than one name. Each alias name has a corresponding alias entry in the DIT which

contains a pointer to the object entry. It has only one attribute (the attribute is required) which is the alias name, and it must be used as the naming (RDN) attribute.

The aliasObject object class is an auxiliary object class defined by IBM whose superior is the top object class. It is similar to the alias object class, except that its one attribute does not have to be used as the naming attribute because the class is defined as auxiliary.

The organizationalRole object class is a structural class whose superior is the top object class in the class hierarchy. It is used to define entries that represent an organizational role. An organizational role may be filled by a person or a non-human entity. An example of an organization role is *Vice President of Business Operations*.

2.6.3 Security Objects

The security objects can be divided into the following categories:

- Authentication objects (RFC 2256)
  - strongAuthenticationUser
  - userSecurityInformation
  - certificationAuthority
  - certificationAuthority-V2
  - cRLDistributionPoint
  - eUser (IBM-defined)
- Authorization objects (IBM-defined)
  - accessGroup
  - accessRole
- Accounting objects
  - account (RFC 1274)
  - eAccount (IBM-defined)

2.6.3.1 Authentication Objects

All these objects are subclassed (in the class hierarchy) from top except for the certificationAuthority-V2 object class which is subclassed from certificationAuthority.

The strongAuthenticationUser object class is an auxiliary object class used to add strong authentication information to structural objects requiring it. This object class contains the certificates. For example, the person object class optionally allows a user password to be specified. If that person directory entry is used for primary authentication and that authentication needs to be

via certificate, the strongAuthenticationUser object class would be appended to that directory entry.

The userSecurityInformation object class is an auxiliary object class that specifies the (encryption) algorithms supported by the server.

The certificationAuthority object class is an auxiliary object class which is used to define entries for objects which act as Certificate Authorities (CA). It must contain the CA's certificate and its certificate and authority revocation lists and, optionally, its cross certificate lists.

The certificationAuthority-V2 object class is subclassed from the certificationAuthority object class and additionally contains delta revocation lists. This class provides compatibility between LDAP V2 and V3 implementations.

The cRLDistributionPoint object class is a structural object class that contains certificate revocation lists and authority revocation lists.

The eUser object class is a structural object class that is subclassed from top. For white pages applications, the person class and its derivatives are generally sufficient. However, in an enterprise environment, a person may have access to a variety of applications and data on a variety of systems in some authenticated and authorized manner. People may play multiple roles (for example, manager, member of a particular team, and so on) where their role may need to reflect their access to resources.

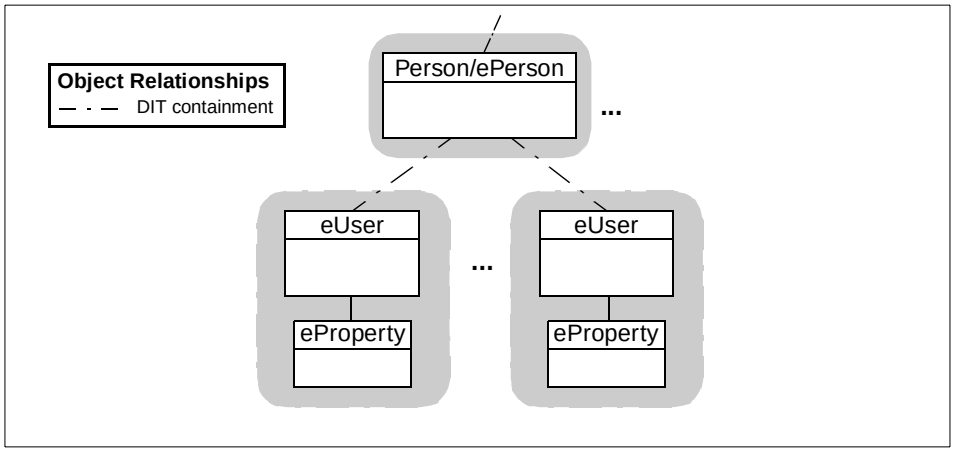


Figure 12. Person - User relationship

The eUser object class may be used to represent this classification and management of their roles and expected application behavior/preferences (Figure 12). The eUser object entry is associated with the person object entry by placing it in the DIT beneath the person entry as shown in Figure 12. (The eProperty objects will be discussed later in 2.6.5, “Profile Objects” on page 55).

2.6.3.2 Authorization Objects

The authorization objects are IBM defined structural object classes whose immediate superior in the class hierarchy is the class top. They provide the definition for directory entries by which the ACL (Access Control List) manager as part of the directory service can grant or deny the requested access. These object classes are associated with IBM ACL model (see 5.6, “Access Control” on page 120). The access-ID (for example a person’s DN or a device’s DN), access group DN, or role DN are part of the ACL entry of an object.

The accessGroup object class is identical in format to the groupOfNames object class, except its objectClass attribute is different (that is, the accessGroup object has an objectClass attribute of accessGroup). IBM defines a new group specifically to hold a list of DNs (human and non-human) because not all the groups are appropriate to be used for access control (for example, an e-mail distribution list is not appropriate to use for access control).

The accessRole object class is also identical in format to the groupOfNames object with the same exception as noted above. Although the definition is the same, the semantic of a role is different from that of a group. A role has an implied set of expectations. An example of a role might be *Administrator*.

2.6.3.3 Accounting Objects

The account object class is a structural object that is subclassed from top and defines the basic information for an account.

The eAccount object class is a structural object class that is subclassed from account and used to define entries that model accounts based on IBM needs. ePerson object entries (and other principals such as eUser entries) have DN assertions to any associated eAccount object entries, but the DN relationship is defined by the account administrator in the eAccount object. The eAccount class may be subclassed by products as necessary to add product-specific information. A query for eAccount objects will yield these application system tailored accounts as well as any additional eAccount subclasses defined by the directory implementor.

Figure 13 on page 54 shows the associations between persons, accounts, and their application systems (the eApplicationSystem class is defined later in this document, but, in short, it represents a logical system, that is, a set of logical and physical resources within a single administrative domain), as well as access control lists and access groups or roles. The account objects represent the authentication relationship between a system and a user (or other principal); they are associated with their application system (or operating system) in the name hierarchy. The DN relationship must be checked whenever traversing the directory to an eAccount object entry based on a principal's back pointer.

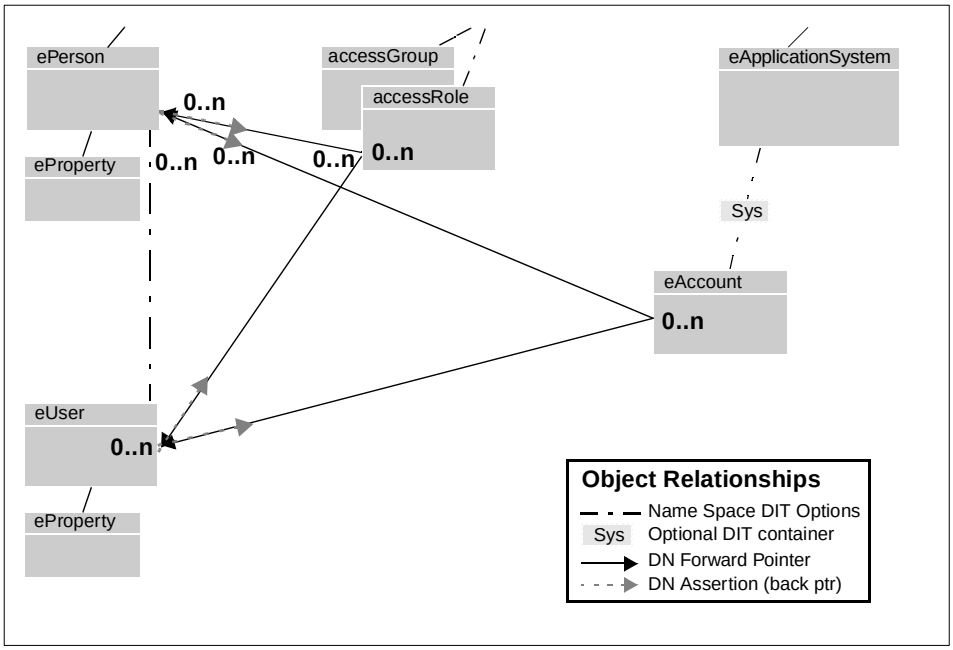


Figure 13. Person - Account and their application system relationships

2.6.4 Policy Objects

Policy objects define a set of common base policies for use in the directory service from which more specific policies may be subclassed. Policies may be security and non-security related.

- passwordPolicy

The passwordPolicy object describes the characteristics needed to implement a policy for passwords that an organization may need to enforce. It

allows for the definition of characteristics such as composition of passwords, aging of passwords, and password reuse.

Policy objects for Service Level Agreement and Virtual Private Network are currently under definition at IETF. IBM is participating in that work.

2.6.5 Profile Objects

The profile objects defined to-date are based on the CIM_Configuration and CIM_Setting classes:

- cimConfiguration
- cimSetting
- eProperty
- PropertySet

They provide the customization attributes for a set of preferences, for example, for a product's shipped defaults, an installation's defaults, or even a user's defaults. These preferences are defined by an eProperty object. In some cases, it is desirable to define a set of defaults, for example, for an organization, but also allow the users in that organization to override some of that organization's defaults to meet their needs. The ePropertySet object allows a hierarchy of preferences to be defined.

The eProperty object class is a structural object class subclassed from the abstract class cimSetting. The eProperty object can be used just about anywhere in the namespace (see, for example, Figure 8 on page 42) to provide a set of preferences for the object to which it is associated. Direct association of an eProperty object to the object for which those preferences are to be associated is via the DIT structure, that is, an eProperty object is placed hierarchically underneath the object in the DIT for which those preferences are to be associated. The eProperty object class has only one required attribute, its name. Its optional attributes include semantic typing information for the eProperty object entry and three paired sets of attributes for describing the type and property values in binary, caseExactString, and caseIgnoreString syntax.

The ePropertySet object class is a structural object class that is subclassed from cimConfiguration. It provides a method for indirectly associating a set of properties with a resource and a mechanism for organizing those sets of preference properties into a hierarchy for coalescing multiple eProperty objects. The association between the property set and the object entries for which it is used may be encoded as a DN pointer in either direction (and the opposite pointer used as a hint). The property set hierarchy (for example, organization defaults and role-based defaults) is defined by linking together

ePropertySet objects using DN pointers to form the relationship. Figure 14 on page 56 shows such an example.

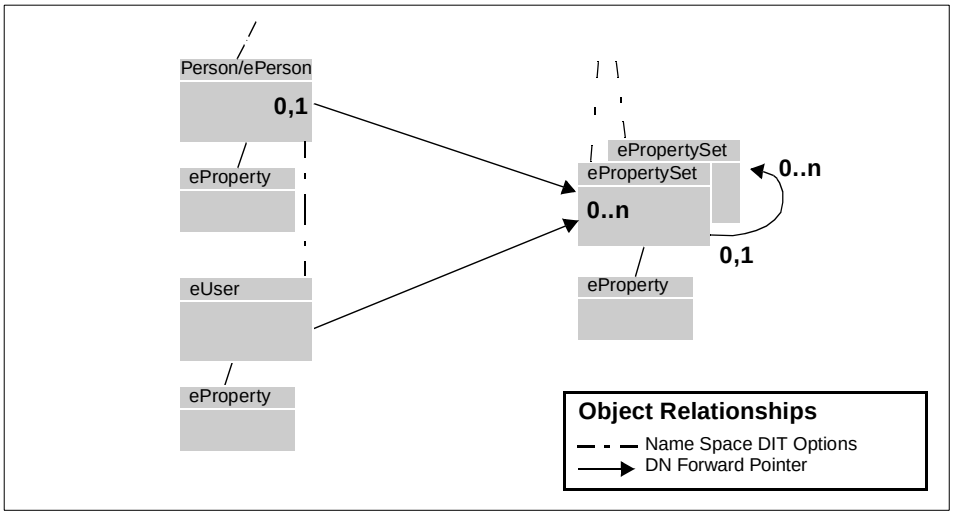


Figure 14. Property set and property object relationships

The model for preferences also permits a hierarchical DIT structure for eProperty objects, i.e. eProperty objects can be containers for other eProperty objects.

2.6.5.1 How to Structure Preferences

Let's look at two examples of how you might structure preferences in the namespace.

Example 1. The distinguished name of an eProperty object might be:

```
cn=1-2-3, cn=Lotus, ou=Raleigh, ou=us, o=IBM
```

where *Lotus* and *1-2-3* are the names of hierarchically-related eProperty object entries. Figure 15 on page 57 shows the DN pointer and namespace DIT subtree relationships for this example.

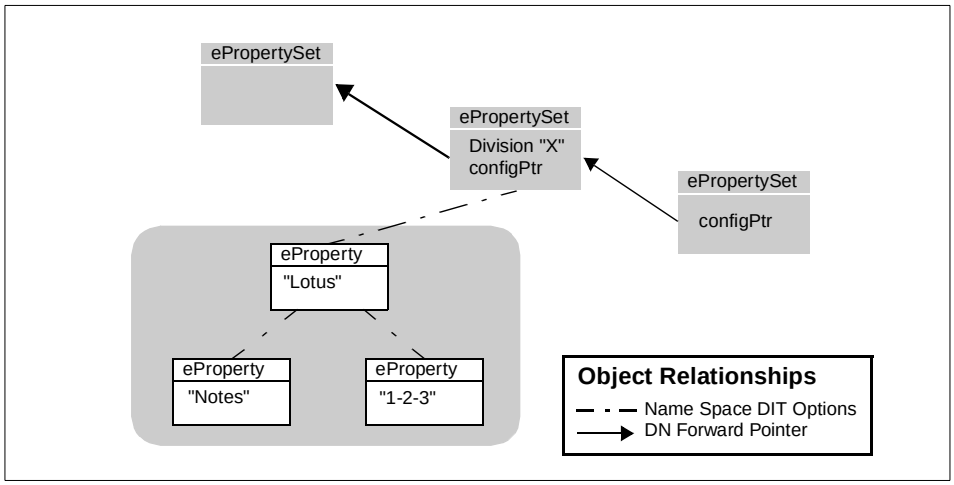


Figure 15. Preferences - Example 1

Similarly, when a user customizes his or her preference, the eProperty object entries may be structured in the namespace subtree of the ePerson or eUser directory entries.

Example 2. This example produces the same result as in example 1 above, but removes empty containers. Removing these empty containers results in a flattened namespace for preferences. This type of structure may improve performance by a small amount. If the *Lotus* eProperty object (in Figure 15) has no property values, it would not appear in the directory. Instead, eProperty objects would appear with RDN values like *Lotus.Text.format*, *Lotus.Notes*, and *Lotus.Components.Spell.V4* where those RDN values carry a hierarchical semantic for preferences. Figure 16 on page 58 depicts this solution.

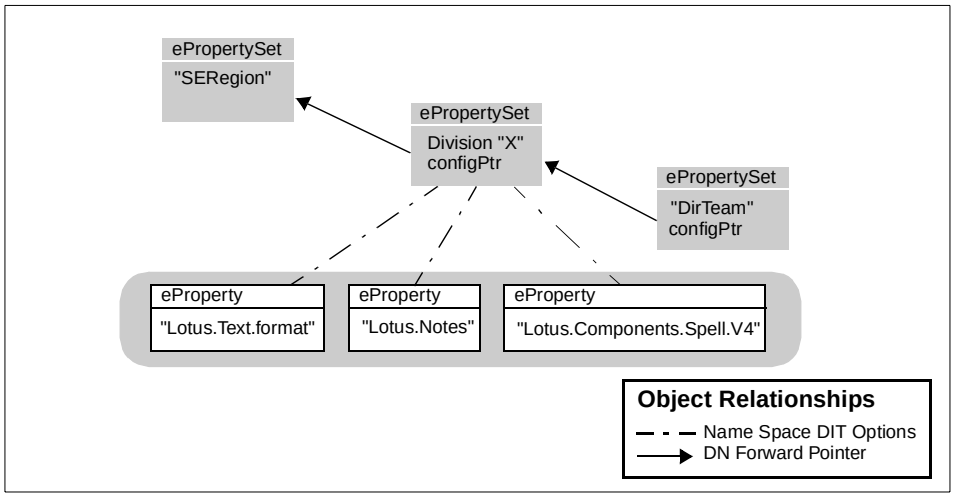


Figure 16. Preferences - Example 2

2.6.6 System Objects

The object classes used for representing systems are derived from the DMTF Common Information Model. Some X.500-derived classes are provided for completeness, but the CIM-derived object classes are recommended to be used for consistency with the rest of the model.

- device (RFC 2256 & X.521)
- labeledURIObject (RFC 2079)
- cimManagedElement (Proposed CIM)
- cimManagedSystemElement (CIM)
- cimLogicalElement (CIM)
- eSystem (CIM)
- eOperatingSystem (CIM)
- eComputerSystem (CIM)
- eApplicationSystem (CIM)

The device object class is a structural object class that is defined by X.500 and is used to define entries representing devices. A device is a physical unit which can communicate, such as a modem, printer, and so on. It is suggested that at least one of the localityName, serialNumber, or owner attributes should be included. The choice is dependent on device type.

The labeledURIObject class is an auxiliary object class subclassed from top and may contain the labeledURI attribute. The intent is that this object class can be added to existing directory objects to allow for inclusion of URI values.

This approach does not preclude including the labeledURI attribute type directly in other object classes as appropriate.

The CIM-derived system object classes are derived from the superclass cimManagedElement, which is based on the proposed CIM_ManangedElement abstract class for user classes. The cimManagedElement class contributes common name, caption, and description attributes to the CIM-derived object classes. The cimManagedSystemElement abstract class adds an installation date and a DN pointer that can be used to associate a derived object entry with configuration information. The cimLogicalElement abstract class adds no attributes to the inheritance hierarchy, but it divides the logical object classes from those that represent physical resources in the CIM model.

The CIM-based classes for systems are derived from cimLogicalSystem. The eSystem object class and its descendents, eOperatingSystem, eComputerSystem and eApplicationSystem, are used to represent their respective systems. The eComputerSystem and eOperatingSystem object classes are straightforward. The eApplicationSystem class provides a collection point for a set of application services and represents a logical system (that is, a set of logical and physical resources) within a single administrative domain. So, for example, an eApplicationSystem entry could represent an administrative set of application resources available on a CICS subsystem, a set of Notes applications, and databases or an Orion domain.

Generally, the directory object entry represents the administrative scope of the application subsystem. The eSystem object class generally will not be instantiated in the directory. However, it can be used as a container-like object entry to represent system administrative scopes within the containing system hierarchy. In the example shown in Figure 17 on page 60, within the scope of an eApplicationSystem DIT, an eSystem object can be used to isolate the creation and maintenance of superuser accounts (for example, a subclass of eAccount) from the creation and maintenance of regular user accounts (represented as eAccount objects in this example).

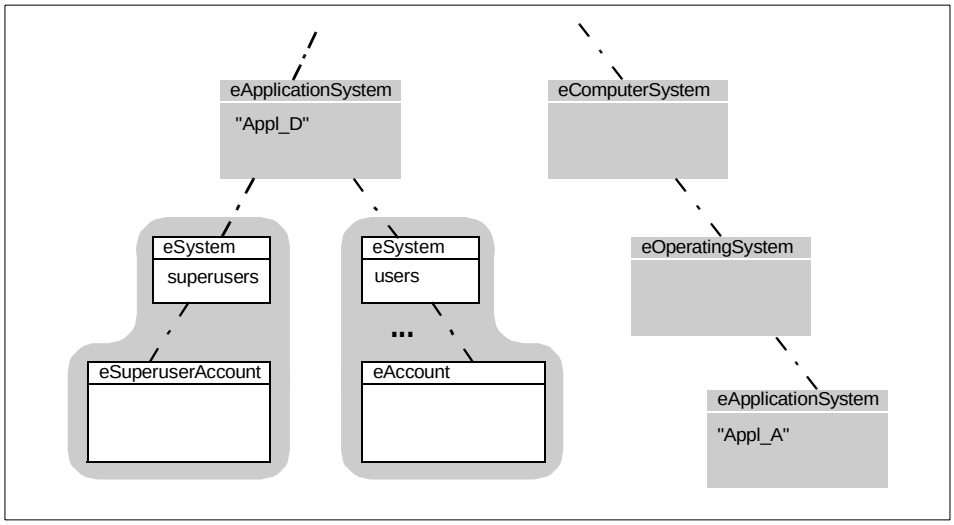


Figure 17. System-level object relationships

The DIT structure for these system-level objects is flexible, but, in most cases, an eComputerSystem object entry will be the superior for an eOperatingSystem entry. An eApplicationSystem object may be used to represent an application subsystem which may be contained within a computer system. In this case, a natural representation is to have the eApplicationSystem entry subordinate to the eOperatingSystem entry representing its execution environment as shown in Figure 17 for *Appl_A*. However, distributed application systems span computer systems, and the administration of those distributed resources may be represented in the directory and in the directory information tree independently of a computer system.

2.6.7 Software Objects

The object classes used for representing software are derived from the DMTF Common Information Model. Some classes derived from X.500 classes (that is applicationProcess, and applicationEntry) are provided for completeness, but the CIM-derived object classes are recommended for consistency with the rest of the model.

- cimProduct (CIM)
- eApplicationSoftware (CIM)
- eSoftware (CIM)
- applicationProcess (RFC 2256 and X.521)
- applicationEntity (RFC 2256 and X.521)

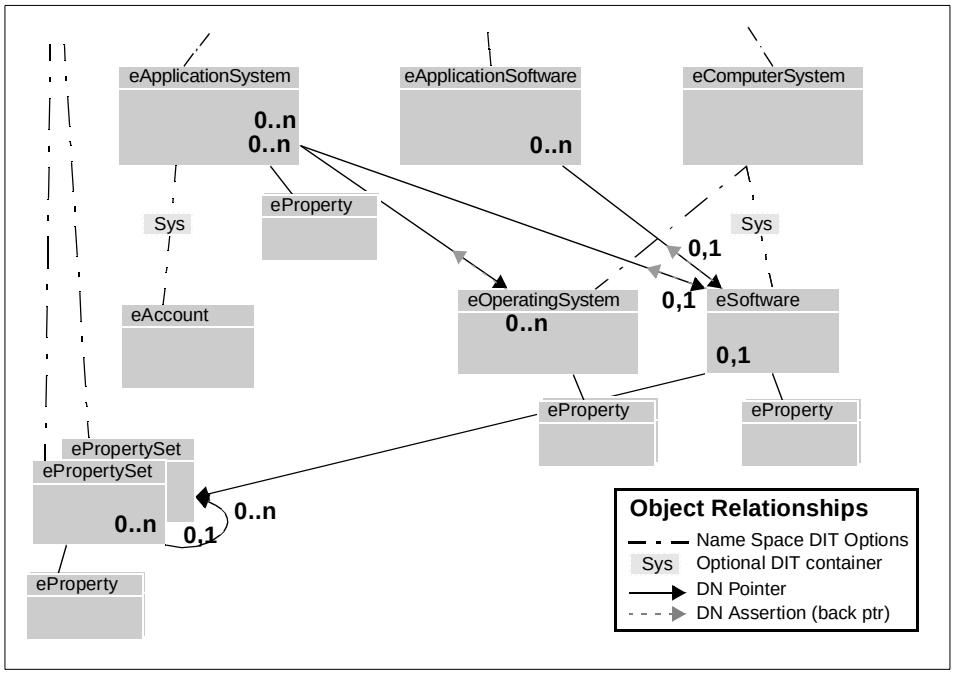


Figure 18. Software object class relationships

Application software, for example, the software installed on a computer system, is represented using the eSoftware structural class (based on both the CIM_SoftwareFeature and CIM_SoftwareElement structural classes). It identifies a product function set (such as, for example, Lotus 1-2-3 bean or an Orion configuration servlet) hosted on a computer system (Figure 18). The eSoftware attributes contain attributes based on the CIM_SoftwareFeature class that identify the vendor, product, and version information as well as the de-normalized CIM_SoftwareElement class attributes, such as the target operating system, language edition, and software installation state. Rather than base the file description information on the entire CIM file system subschema, IBM includes file information in the eSoftware class as well.

Because there can be multiple computer systems with identical copies of a software product, the model also adds an eApplicationSoftware structural object class (based on the CIM_Product class and derived from the cimProduct directory object class) to represent the entire set of instantiated copies of the software. The eApplicationSoftware entries may be used as meta-objects to permit the selection of an instance of the software based on network policies and client location information. In the near-term, products may choose a combination of heuristics such as physical location,

longest-prefix IP address matching, load balancing weights, and so on. The eApplicationSoftware and eSoftware classes model software in various stages of installation (that are deployable, installable, executable and running), but the running state is not modeled for most software.

The applicationProcess object class is a structural object class that is used to define entries representing application processes. An application process is an element within a real open system which performs the information processing for a particular application. It is defined by X.500.

The applicationEntity object class is a structural object class that is used to define entries representing application entities. An application entity consists of those aspects of an application process pertinent to OSI. It is defined by X.500.

2.6.8 Service Objects

The service and service access point object classes are derived from CIM.

- cimService (CIM)
- eService (CIM)
- eSAP (CIM)

The eService structural object class (based on the CIM_Service class) provides the directory representation for the long-running software services (for example, a Web service, a configuration service, or an authentication service). An eService object entry may be used to represent an accessible service. For example, the equivalent instances of an SQL server, where the externals (interfaces, access rights, and data) are the same, might be represented as a service. The eService entries differ from the eApplicationSoftware and eSoftware entries. Whereas eApplicationSoftware and eSoftware entries present programs and supporting files, an eService entry that is started represents a set of running images with, for example, ports on which they listen for incoming connections. As with the eApplicationSoftware class, the eService class permits the selection of an instance based on network policies and client location information. In the near-term, products may choose a combination of heuristics such as physical location, longest-prefix IP address matching, load balancing weights, and so on.

The eSAP structural object class (based on CIM_ServiceAccessPoint) represents those service connection points as instantiated on a computer system with a URL for access. Both eService and eSAP object classes may be extended with product-specific subclasses as needed.

Figure 19 shows the relationship between the services object classes.

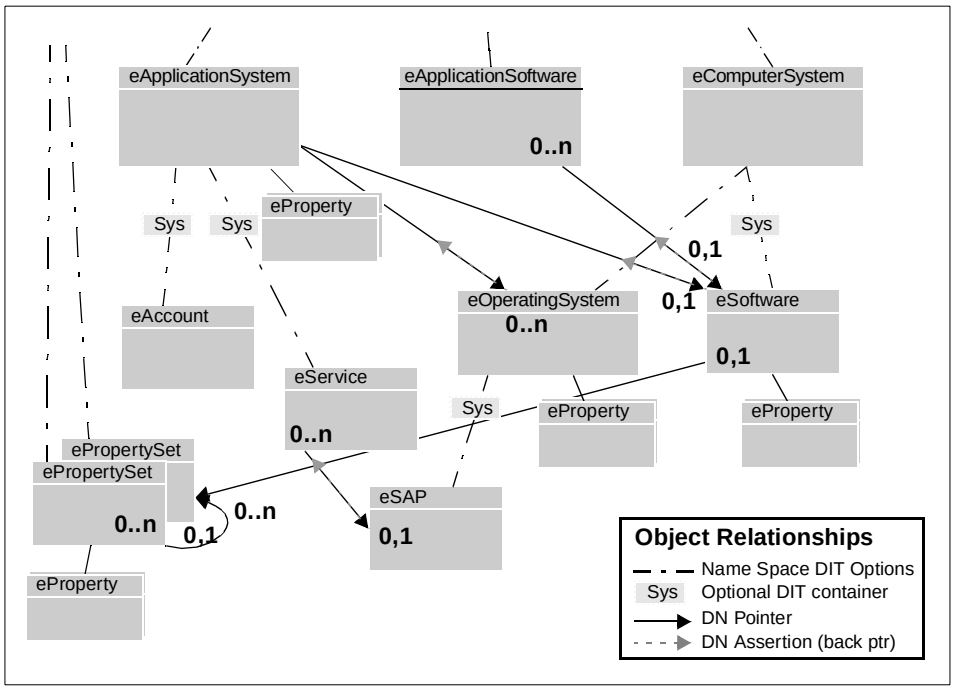


Figure 19. Service object class relationships

2.6.9 Other Objects

This is a collection of objects that are IBM-defined and used by the exploiting products. These objects will be categorized at a later time.

- publisher

The publisher object class is an auxiliary object class that is subclassed from top. It contains the name of the host publishing the information in the structural class to which this auxiliary publisher class is attached.

2.7 The IBM Schema Repository

For its internal product development, IBM uses a tool for directory exploiters (either product or application developers). The IBM Schema Repository is a single place that holds all the schema definitions that IBM ships with its IBM SecureWay Directory. This is the central location for schema definitions that the IBM SecureWay Directory supports, and it also incorporates applicable schemas from standards organizations including IETF, DEN, and DMTF/CIM.

The benefits of such a centralized repository are:

- It eliminates redundancy in object class and attribute definitions.
- It encourages reuse of defined object classes and attributes.
- It encourages exploitation of the directory.
- It provides information needed to work with industry groups defining schema.

This information is kept on-line through the use of a Lotus Domino database. This database provides:

- object classes and attributes of the IBM schema
- forms to define new object classes and attributes with OID assignments
- views of the object classes and attributes already defined along with their state information (requested, active/approved, retired)
- tools for automated creation of the schema files

IBM employees can access the database by pointing their browsers at:

`http://ends390d1.endicott.ibm.com/ldap/oidrgy.nsf`

or using their Lotus Notes client, where the server name is ends390d1/ends390 and the directory/filename is LDAP/oidrgy.nsf.

IBMers may also replicate a copy of the database to their workstations for their own purposes.

Non-IBM readers may find this information useful for a few reasons. They may consider implementing a similar function within their own organization when starting to build a new directory infrastructure based on LDAP. They may also want to define additional object classes and attributes needed in their own systems and applications that are not covered by the schema provided by the IBM SecureWay Directory. The benefits will be similar as described earlier in this section. Contact your local IBM representative to find out what is the best way to get the described tool (or similar) implemented in your own environment.

Chapter 3. A Step-by-Step Approach for Directory Implementation

In the previous two chapters, we have introduced the principles of an LDAP-based directory service and the possible implications for LDAP implementation projects. Now that directory standards are emerging, the opportunity exists for enterprises to simplify their directory problems. If an enterprise is able to store directory information once and administer it in one place, the total cost of maintaining this information is significantly reduced.

This chapter provides an overview of the planning and design work required for most LDAP implementation projects. Key concepts are introduced, and the subsequent chapters will have more detailed information about actual implementation. Though the descriptions in this chapter are not very detailed, and some of the steps outlined below may seem trivial and therefore tempting to be bypassed, it is strongly suggested, however, to follow each step as explained below. The time you spend on trivial issues, such as defining your data, will be worthwhile.

Designing a directory has much in common with designing a relational database. Distinctively, a directory design cannot be verified by applying checking rules, such as the normal forms that apply for relational databases. Thus, designing a directory allows for more freedom that can be used for specific requirements, such as application requirements, performance considerations, or namespace partitioning. It must be clearly understood, however, that common sense as well as fundamental application and database design principles apply equally for a directory. The most important principles to have in mind when designing a directory are:

- Avoid data redundancy
- Ensure data consistency
- Cross-platform and cross-application exploitation
- Extensibility
- Use standards whenever possible

The step-by-step approach explained in the following sections give you a guideline for designing a directory.

3.1 Define the Objective for Using an LDAP Directory Service

There have to be clearly defined business requirements before you start your directory service project. What is meant by a business requirement is that your business is driving you to exploit new technologies like LDAP into the IT

environment. Such a requirement can be the implementation of a new extranet application to link your suppliers with your legacy systems, and you need to integrate this group of new users with your existing authentication and access control methods. Or, it can be a consolidation of multiple dissimilar directories to a common directory to simplify administration and, thus, reduce the related costs.

Define the objectives for the directory service implementation project based on your requirements. The objectives should define the service that you are planning to implement including the goals you plan to achieve with this new service, who will be the users (applications/users), what will be the availability requirements for the directory service, and so forth.

This first step is completely independent of any directory implementation.

3.2 Define the Data to Store in the Directory Service

Planning the directory's data is the most important aspect of the directory planning activities, and it is probably the most time-consuming aspect as well. Based on the requirements defined in the first step (see above), you should locate all the data stores where the relevant directory information is managed. As this survey is performed, expect to find that some kinds of data are not well managed; some processes may be inefficient, inadequate, or nonexistent altogether; and some kinds of data may not be available at all. All of these issues should be addressed before finishing a data-planning phase.

3.2.1 The Type and Use of Directory Data

A directory service is not a substitute for a generalized relational database or transaction system. In a directory service, reading or searching data is generally more frequent than updating the information. Some of the common criteria for storing information in the directory include:

- The data will be accessed by multiple applications.
- The data will be accessed from multiple locations in the network.
- The data to be stored conforms to an industry or open-system definition that has been standardized on schema and namespace definitions.
- The data is typically more often read than written (updated).

Some applications use the directory for reading data only and will not store any data in the directory. This type of application typically uses the existing schema and namespace definitions for developing their LDAP code. Applications that write to the directory have additional design considerations. Thus, evaluate how the data is used by your application. For example, if the

data is used during the initial start-up of the application prior to the application's ability to make a call to the directory, it is not a good candidate for storing in the directory for obvious reasons.

3.2.2 Survey the Directory Data

Having in mind the types of data suitable and unsuitable for use in a directory, it is now possible to survey what the directory service data will be. In doing this, it may be helpful to do the following:

- Survey the organization and identify where the data comes from (such as Windows NT or Novell NetWare directories, Human Resources databases, e-mail systems, and so forth).
- Determine what directory-enabled applications to deploy and what their data needs are. Do not forget possible future applications.
- Determine who needs access to the data, particularly the organization's mission-critical applications. Find out if those applications can directly access and/or update the directory.
- For each piece of data, determine the location where it will be mastered, who owns the data, that is, who is responsible for ensuring that the data is up-to-date.
- If data is going to be imported from other sources, develop a strategy for both bulk imports and incremental updates. Try to limit the number of applications that can change the data. Doing this will help ensure the data integrity while reducing the organization's administration.
- Identify duplications and data that is not actually used or required. Harmonize the data by eliminating such duplications, and discard unnecessary data.

Having decided on the type of data to use in the directory service, what the directory will be used for, and how the data will be updated, it is possible to start structuring the data.

Note on Updates

It is considered a general design guideline that data that is subject to frequent updates should not be stored in a directory because directory services are optimized for queries. The IBM SecureWay Directory with the relational database backing store, however, does provide surprisingly good update performance. The decision on whether or not to store frequently-changing data in a directory should also be based on application requirements and client cacheing capabilities and algorithms.

3.3 Evaluate Data and Its Relationship to Directory Schema

Rather than defining the data as if it is the only data in the directory and the application the sole application, the design point is to evaluate the data you defined in the previous step against the default schema defined for the IBM SecureWay Directory (see Chapter 2, “Schema and Namespace” on page 31). The design objective for directory exploitation projects is to maximize the sharing of common objects and attributes. For each piece of data, determine the name of the attribute(s) that you will use to represent the data in the directory and the object class(es), that is, the type of entry, that the data will be stored on. Wherever possible, you should use existing schema for your application rather than private definitions.

It is very likely that using existing objects and attributes will have an impact on the design of your application as you re-engineer from a side-file-based (data defined and used by just one application) to a shared directory. The value of an integrated directory is the reason for each application to make the changes necessary to use common schema definitions.

Most LDAP-enabled application clients are designed to work with a specific well-defined schema. Shrink-wrapped standard applications (and any tools and applications that depend on standard applications) most likely only work with a standard schema. This is an important reason why LDAP-based directory services should support at least the standard LDAP schema. Then, the schema may be extended as the site discovers site-specific needs not met by the standard schema.

If your application data is not found in the standard schema, new objects and/or new attributes can be added. The entire process for evaluating the use of schema and the need for extensions, as explained in Chapter 2, “Schema and Namespace” on page 31, will ensure that extensions to the schema are validated and implemented correctly.

3.4 Define and Assign Responsibilities for the Data

As soon as the data that goes in the directory has been evaluated (or, possibly, during the process of data evaluation), responsibilities need to be defined and assigned to the appropriate persons or administrative authorities, such as departments in a company. These authorities must be involved early in the implementation process, and they will keep these responsibilities for as long as *their* data is in the directory. In particular, the responsibilities are:

- To ensure that all necessary data for the area of responsibility is taken care of in the design phase and that no unnecessary duplications exist

- During the migration phase, to ensure that possible problems, for example, due to character set conversion difficulties, are promptly detected and corrected
- To reliably maintain the data after the directory has been put into production

3.5 Evaluate Data and Its Placement in the Namespace

Every LDAP directory will implement a structure, usually hierarchical, that defines the namespace. The namespace definition will determine where you should place your objects and attributes. This will then determine the distinguished name (DN) of your entries. If your application data does not *fit* into the existing namespace and you have considered possible adoptions to the existing namespace with no success, the namespace will need to be extended.

It is important that namespace extensions are coordinated within your organization to avoid any duplication and inconsistencies, just as IP addresses and hostnames need to be coordinated within an organization.

When deciding on suffixes, where a suffix is the root DN of a directory tree as described in section 2.3, “Directory Information Tree (Namespace) Structure” on page 36, one of the methods is to use the same naming structure for LDAP as is used for X.500. This method will set the root of the directory tree to a specific organization in a specific country or to a specific organization and organizational unit.

Another method that you can use, if the X.500 does not seem appropriate, is to use the DNS naming model when choosing the directory suffix. This would result in a suffix using the domainComponent (dc) attribute, for example <dc=xyz.se, dc=abc.us, or dc=abc.com>.

Choosing to branch a directory tree based on the organizational structure, such as departments, can lead to a large administrative overhead if the organization is very dynamic and changes often. On the other hand, branching the tree based on geography may restrict the ability to reflect information about the organizational structure. A branching methodology that is flexible and that still reflects enough information about the organization must be created.

3.6 Evaluate the Existing Security Policy

If, based on the evaluation done in section 3.3 on page 68, you determine that you will use (reuse) existing objects and attributes in the LDAP directory, you must re-evaluate your security policy against the existing security policy for those shared objects and attributes that are to be stored in the directory.

Having designed the directory tree, you now need to decide on a security policy. A security policy should be strong enough to prevent sensitive information from being modified or retrieved by unauthorized users while simple enough to keep administration simple and enable authorized parties to access it easily. Ease of administration is very important when it comes to designing a security policy. An overly-complex security policy can lead to mistakes that either prevent people from accessing information that they should have access to or allow people to modify or retrieve directory information that they should not have access to.

The security policy that needs to be designed for the directory service is a reflection of the:

- Kind of data (confidentiality of information that will be stored in the directory)
- Ways in which clients will be accessing the directory
- Ways which will be used to update and manage the directory
- Acceptable administration effort for security

To reach these goals, two basic areas must be considered and the following questions must be answered: What level of security is needed when clients identify themselves to the directory server, and what methodology will be used when authorizing access to the different kinds of information in the directory?

Security aspects are discussed in Chapter 5, “Directory Security” on page 103.

3.7 Define the Migration Model

Having done all the planning and design work to introduce LDAP as a new directory service, one more important task has to follow. Remember that the job of designing an LDAP directory very likely started with an analysis of currently available data and the various locations and services where directory data might be stored and serviced from. For example, there might be several LAN-based directories installed and operational already in your

organization, such as IBM Warp Server domains, Microsoft Windows NT/95 domains, Novell NetWare domains, or other databases with directory-type information.

In most cases, introducing LDAP will not be an introduction of a new service from scratch, but some kind of a migration from existing services to LDAP will have to take place. Once your organization has decided to move to an open, vendor-independent directory service, which describes LDAP perfectly, you have to carefully analyze if and how current directories can participate in, benefit from, or even be migrated to an LDAP directory infrastructure. Bear in mind that such a migration is not just a software or application replacement; it is also an opportunity and the right time to redesign and harmonize the data in your directories.

Because migrating current directory services to LDAP may not be a trivial or risk-free undertaking, your decision might be to run and maintain proprietary directory services in parallel with LDAP for some period of time (while this is certainly not textbook advice, it may be considered for practical reasons). This might also be necessary when you need to keep some of the proprietary services that cannot be supported with LDAP running for some time. Another common reason for running LDAP and other directory services in parallel might be a shortage of skills or staff personnel.

The migration approach just described involves little risk because there is always a backout method in case a migration step fails. Clients may be migrated within a time period, which is especially useful in large installations. New services may experience a delay when problems are encountered after their introduction. Migration of a complete service at one time imposes a much higher risk because both servers and clients (including applications) have to be ready immediately. The only (usual) backout solution is to go back and (re)install the old environment. The decision on which scenario to follow largely depends on the size of the installation, the risk that can be taken, and the possibilities to run directory services in parallel.

If planning for a migration, it should also be considered that, according to their public program announcements, vendor-specific directory services, such as Novell's NDS or Microsoft's Active Directory, support an LDAP interface. Using this interface, LDAP clients can access data in these directories.

Different migration models are described in detail in 4.6, “Migration from the Previous Release” on page 91 and in 4.7, “Migration from Non-LDAP Sources” on page 92.

3.8 Define the LDAP Programming Model

This step is the first time we actually discuss LDAP programming. The initial six steps are focused primarily on data. This is a good indication that directory exploitation is all about data: what data to store, how to store it, where to store it, and how to share it. One of the reasons LDAP has been so well received is that it is a lightweight client in a client/server computing model.

But, like LDAP programming in general, the client/server implementation has also been oversimplified in some of the literature. The programming examples usually describe a client application as invoked by an end user using the LDAP client that then communicates, via LDAP protocol flows over TCP/IP, to an LDAP server. Using the ever-present address book application as the norm, the two-tier client/server computing is defined as the *standard* implementation.

In practice, the prevalent programming model for directory exploitation projects is usually a three- (logical) tier model where the end user interacts with an application or subsystem, and that application or subsystem then accesses the LDAP directory. The second tier uses the client code located on the second-tier system to access the directory.

This three-tier computing model is worth mentioning for two reasons:

1. The two-tier client/server model usually describes the interaction with end users where the end user *signs-on* (binds) to the directory server with his/her DN (distinguished name). The assumption that the end users know or have their DNs and that they can then be used to authenticate to the directory server may not be true in the three-tier model. The use of an LDAP directory server by the second tier may be transparent to the end user, or an application that has been re-engineered to store side file information in the LDAP directory may not want to (or be able to) change existing end user business applications. In this case, your design may have to be able to map between the current identification mechanism (often eight-character user ID) and the DN of that person in the LDAP directory. This is called delegation and is further detailed in 5.5, “Delegation Model” on page 117.
2. The two-tier model that defines the interaction as being between the end user and the LDAP directory may be invalid for your three-tier application, either because the second-tier application accesses the LDAP directory using its own identity to perform some function or because the second-tier application accesses the LDAP directory to read the end user’s record in

order to do its own authentication rather than rely on the authentication capability of the LDAP server.

The application development considerations are explained in more detail in Chapter 8, “Developing Directory-Enabled Applications” on page 223.

3.9 Define the Deployment and Performance Criteria

The physical design of a directory involves building a network and server infrastructures to support availability, scalability, and manageability. Methods to do this in LDAP are partitioning (referrals) and replication (replication is actually not standardized in LDAP Version 3, but most vendors, including IBM, do have an implementation). In this section, we concentrate on deployment issues regarding when partitioning and/or replication is appropriate when trying to reach the goals of availability, scalability, and manageability, and what the trade-offs are.

Availability for a directory service may not be a hot issue in cases where the directory is not business-critical. However, if the use of the service becomes mission-critical, the need to design a highly available system emerges. Designing a highly available system involves more than what is supported by the LDAP standards. The components from LDAP that are needed are partitioning and replication. Since high availability involves eliminating single points of failure or reducing their impact, it is necessary to have redundant hardware, software, and networks to spread and lower the risk.

3.9.1 Availability of the Directory Service

A simple approach to create a highly available directory service (despite using application-independent solutions with high-availability hardware and/or software implementations) is to create a master and a slave directory server, each one on its own physical machine. By replicating the data, we have eliminated the single point of failure for both hardware and software failures. A mechanism must be added to handle client redirection if one server fails. This can be done manually or semi-automatically by a DNS switch over, or automatically with a load-balancing technique by using a router designed for this (such as the IBM eNetwork Dispatcher). Such a router forwards client requests to one of the servers based on configurable criteria. There is also the issue of network bandwidth and its reliability to take into consideration. In some cases, it may be necessary to distribute a replica to another network with slow network connections to the master.

If the method of spreading the risk is used to create high availability, it is possible to partition the directory tree and to distribute it to different locations,

LANs, or departments. As a side-effect, depending on how the directory tree is branched and distributed to these servers, each location, department, or LAN administrator can then easily manage his/her own part of the directory tree on a local machine if this is a requirement. If a single server fails in such a configuration, only a portion of the whole directory will be affected.

A combination of the methods explained above can be used to create a dynamic, distributed, highly available directory service.

Replication and referrals are further explained in 7.6, “Replication” on page 178 and 7.7, “Referrals” on page 184.

3.9.2 Performance Considerations

An application written to use LDAP will exhibit certain performance characteristics based on the type, complexity, and frequency of accesses to the directory. This is an important consideration, especially for those products that are moving from using side-files to using a client/server directory service.

Another aspect of performance is the location of the directory (or directories) in the network. While the IBM SecureWay Directory will run on and integrate with all IBM and other systems, this means that the directory *may* be installed on each system but not that it must. A directory-enabled application may exhibit better performance by locating the IBM SecureWay Directory on the same system, but this advantage may be more than discounted by the administrative burden of maintaining dozens or hundreds of distributed directories in an enterprise. The number and location of IBM SecureWay Directory servers (master and replicas) in an enterprise is a balancing act that includes:

- The overall size of the namespace
- How/if to split the namespace (suffix) across the enterprise
- Administration and administrative sub-domain considerations
- Processor capacity and throughput
- Network bandwidth and load
- Performance requirements of directory-enabled applications

The deployment and performance considerations are discussed in more detail in the Chapter 4, “Managing an LDAP Directory” on page 77.

3.10 Step-by-Step Summary

As was mentioned at the beginning of this chapter, the step-by-step approach as described here only gives an overview of the planning and design work

required for most LDAP exploitation projects. Key concepts were introduced, and these concepts can be used as a starting point when planning for a common directory service project.

IBM Application Framework for e-business is a model that identifies key elements for developing and deploying e-business applications. Each element is based on open vendor-neutral standards allowing you to substitute components from any vendor that supports these standards. As described in 1.5, “The Framework for Creating Enterprise-Wide Solutions” on page 17, a common directory service is a key element in the network infrastructure. You may find it useful to use this model when modifying your existing infrastructure to include a common directory service. Introducing the new key element into your systems environment may also cause a need for updates in your overall systems architecture. The Framework can serve as a model for this architectural re-design as well.

The step-by-step approach just described is not the only proven methodology for directory exploitation projects but is described here for planning purposes. IBM Global Services has architecture and infrastructure design service offerings where IBM consultants and architects use proven methodologies to execute such engagements. Contact your local IBM representative if you want more information about available offerings.

Chapter 4. Managing an LDAP Directory

Administration of the IBM SecureWay Directory starts with the directory design and involves planning, setup and configuration of the systems, daily operations, and, finally, maintaining the data within the directory.

This chapter describes the management of a directory service as far as design and planning are involved. An overview of administration tools is provided, and methods are shown that can or need to be in place to establish a reliable, corporate-wide directory service using the IBM SecureWay Directory. Subsequent chapters will then provide more detailed information and *how-to* instructions for the specific management topics.

4.1 Overview: Administration Tools, Utilities, and APIs

Depending on specific needs and preferences, administration of the IBM SecureWay Directory can be done using several ways:

- Using the graphical administration tool
- Using the graphical Directory Management Tool (DMT)
- Using command line utilities
- Using the APIs for custom-written utilities
- Using the Tivoli User Administration LDAP Connection

The most obvious (and probably most intuitive) way to administer the IBM SecureWay Directory from a systems management point of view is by using the graphical administration interface that is provided with the product. It allows an administrator to both examine the current status and settings, and to configure and change server settings and certain directory properties. Administrators are discouraged from editing the configuration file(s) with a text editor, and should rather use this administration tool to change the configuration of the IBM SecureWay Directory server. The graphical administration interface, however, does not support the administration of the data stored in the directory.

Administration of the data contained in the directory can conveniently be done using the DMT. It lets an administrator browse and edit not only the data contained in the directory but also schema definitions (object classes and attributes) and the directory tree itself. Search capabilities are provided to help find entries. Although it is not its primary purpose, the DMT can also serve as a perfect tool to learn about the various objects contained in an LDAP directory and their relationships to each other.

If large amounts of data are involved or if automation is a goal, a graphical tool might not be appropriate to use. For this purpose, command line utilities are provided with the IBM SecureWay Directory. These utilities provide several methods of directory administration and data manipulation. Simple and powerful command line utilities have been developed to perform all common LDAP operations. For example, these utilities read in LDIF (LDAP Data Interchange Format) files, translate the file into the format necessary to use the LDAP protocol, and then handle the request(s) with the directory server.

Using the provided utilities is the simplest method of updating a directory's data. It takes little effort to set up. All that an administrator must do is create the appropriate LDIF files. The directory administrator does not need to worry about the sequence of LDAP operations against servers, server versions, binding, referral chasing, and unbinding. Additional facilities for performing the operations over SSL are also provided. The necessary information is passed to the tool on the command line and sometimes through an LDIF file, and the rest of the particulars to complete the operation are taken care of by the utility.

These utilities can be useful for batch operations, such as adding a large number of users at a time or adding a new attribute to many entries. Custom-written administration applications that need to have a customized user interface can develop an interface for modifying directory information, write this information to an LDIF file, and then call the client tools.

As an alternative to using the administration utilities, custom-written administration tools can directly use the LDAP APIs provided with the libldap library. Such administration tools might be desirable when typical data administration, such as adding or modifying employee data, is done by non-technical staff. Writing directly to the API layer may also be necessary for applications that need to control the bind/unbind sequence, or, perhaps, want to customize the referral behavior. This is a more difficult approach since the developer must deal with the conversion of the data to the structures that are sent over the LDAP protocol. Additionally, the developer must be aware of a particular security setup, such as SSL. The C programming language API is described further in 8.2, “C LDAP Application Programming Interface (API)” on page 235.

Another approach for custom-written tools is to use the Java Naming and Directory Interface (JNDI) client APIs. This approach is good for developers who must maintain their client applications across multiple platforms. The JNDI interface is easy use and provides a means of quickly developing

portable programs using the LDAP directory. More about the JNDI can be found in 8.1, “Java Naming and Directory Interface (JNDI)” on page 223.

The Tivoli User Administration LDAP Connection, as introduced in 1.7.4, “Tivoli User Administration: LDAP Connection” on page 26, is another way to manage directory data. It is, however, limited to managing users to the extend Tivoli supports. More about the LDAP Connection can be found in 4.4, “Tivoli TME Considerations” on page 85.

4.2 Centralized versus Distributed Administration

The directory administrator (the user with the *root DN*) is, by default, the only one person who can administer information in the directory. At times, it will be necessary to allow other users to have administrative privileges on all or portions of the directory. The Directory Information Tree (DIT) can be divided into administrative areas; the directory administrator can give other distinguished names (DNs) full privileges to manage some subsection of the directory. In order to grant a user administrative permission to a subtree, that user DN must be specified in the entry owner attribute. The administrative domain will be delimited by the value of an owner inheritance attribute (OwnerPropagate); if it is set to FALSE, the scope of the administrator will be the single entry on which the owner was set, and if OwnerPropagate is set to TRUE, the administrative domain will be the entire subtree unless a new entry owner is specified in a descendant entry. For more details, see 7.9.3, “Ownership and Access Control” on page 206.

The following three sections discuss some general consideration that apply when administration is being distributed to multiple administrators.

4.2.1 Who Administers The Data?

The simplest model of data administration is to allow only the directory administrator (root DN) to administer all the data. While this is clear cut and simple, it is not probable that the same person will be administrating all applications using the directory. In order to facilitate data administration, it is possible to allow additional users to administer parts of the directory. This flexibility allows administrative privileges to users, groups of users, or users acting in particular roles.

There are two methods of granting a user the ability to administer directory data. One is to grant them permissions on attribute classes via the ACL entry, and the other option is to make them an entry owner.

By making someone an entry owner, that person becomes an administrator for that particular entry. They can modify any attribute within the entry and can change the access control lists of that entry. Users given permissions by the ACL entry cannot modify the ACLs on the entry and only have permissions to what is specified in the ACL entry.

In order to facilitate administration of the ACL information, two DNs have been created and are used as pseudo-subjects. These pseudo-subjects are not represented by an object in the directory but can be used on an ACL. They are used to refer to large numbers of DNs, which, at bind time, share a common characteristic in relation to either the operation being performed or the object on which the operation is being performed.

The first of these is the group <cn=Anybody>. When specified as part of an ACL, this group refers to all users. Users cannot be removed from this group, and this group cannot be removed from the database. <cn=Anybody> is considered to be the group of all unauthenticated users or any user which does not have a specific ACL on an object. By default, this group has read, search, and compare permissions to attributes within the normal class.

The second pseudo DN is the access-id <cn=self>. When specified as part of an ACL, it grants permissions when the bind DN matches the object DN on which the operation is performed. If an operation is performed on the object <cn=personA, ou=IBM, c=US>, permissions associated with the access ID <cn=self> would be granted when the bind DN is <cn=personA, ou=IBM, c=US>.

An example of when a pseudo subject might be useful is a phone book application. The administrator might want to give individuals permissions to update parts of their own entry. If all the attributes that the user should be able to update are in the normal class, the administrator will put a single ACL on the entry (or propagate some parent ACL) which stated <access-id:cn=self:normal:rwsc>. Each user would now have permission to update parts of their own entry.

You can see an example in 7.9.3, “Ownership and Access Control” on page 206 that gives you more information regarding this topic.

4.2.2 Attribute Grouping

It is likely that many attributes will require the same type of protection. It is, therefore, useful to coarsen the access policy granularity by grouping attributes with similar access sensitivities. This reduces the number of access lists within the directory and greatly simplifies administration.

Attributes are grouped together in *attribute access classes*. Within the schema file, attributes are mapped to an access control class. Each class is discrete, that is, access to one attribute class does not imply access to another class.

Instead of specifying that a subject has access to an attribute, the administrator gives a subject permissions to an access class. This grants the subject the specified permissions to all attributes within that access class.

Let's use the example of modeling an organization. People attributes can be grouped into several categories. One might group *critical* attributes, such as salary, job grade or level, and hire date together. Attributes such as phone, location, and e-mail address will be considered *normal*, and the performance evaluation ratings attribute might be *sensitive*.

Subjects will then be given access to a given class of information. If the information they were requesting is within a class to which they are authorized, then the LDAP operation succeeds. For example, a person might have read and write access to their own normal attributes but only read to sensitive and critical. One might want the person's manager to have read access to normal and critical and read-write access to sensitive while critical attributes can be read and written by anyone who is acting in the personnel role. The ACL for this might read something like (Aclentries are explained in 5.6, “Access Control” on page 120):

```
Aclentry: access-id:cn=self:normal:rwsc:sensitive:rcs:critical:rsc
Aclentry: access-id:cn=manager, ou=org,
c=us:normal:rcs:critical:rsc:sensitive:rwsc
Aclentry: role:cn=personnel, ou=org,c=us:critical:rwsc
```

Note

The preceding example has granted search and compare permissions wherever read permissions are given.

The default schema has classified each attribute into an access class based on the expected use of the attributes. The assigned classes can be seen by looking at the schema files. When adding attributes to the directory schema, the administrator will have to assign attributes to access classes.

Generally, overall guidelines for establishing the types of data that fall into each of the attribute classes can be decided on by the directory administrator. For instance, normal attributes are those which are readable by everyone, modifiable by the object DN, and possibly modifiable by some other DNs which have the authority to do so. Sensitive attributes might be those that are

writable by the object DN and possibly one or two other DNs. Critical attributes might be those that are readable to the object DN, but writable by one or two other administrative users.

4.2.3 Multiple Applications

When there is only one application that uses a directory, defining the objects and attributes and administering the data in the directory is very simple. Or, if an administrator is simply merging multiple existing directories into a single directory, it's easy to merge the objects and the attributes and decide on a grouping policy.

This activity becomes more difficult as the number of applications and administrators increases. It is very important to maximize common semantics and syntax of similar objects and attributes. However, for some small subset of object classes, multiple applications may need to use the same object and its attributes in slightly different manners.

In an ideal situation, the application administrators will reach an agreement regarding which attributes belong to which access classes. Furthermore, they will come to a consensus as to which DNs are given permissions to administer those classes. While this sounds like an unlikely event at first, it is realizable. For instance, two applications (A and B) want their own attributes, which they alone administer. Previously, both had used the *critical* attribute class for these attributes. If they both continue using the critical attribute class, the people granted rights on the critical class can now update the attributes for both application A and application B. This may, in fact, be the desired scenario if the administrator for the two applications is the same. If not, one must consider that the number of administrators updating critical attributes is probably very small. One alternative is to trust that the administrator for application A will read the documentation and know which attributes belong to their application and operate only on those attributes. Unless the administrator is trying to maliciously undermine another application, there is no worry that they will harm the other application's attributes.

For those instances where an agreement cannot be reached, there is another possibility. Each application can create a child node that contains the attributes necessary for its own function. For instance, the object in question might be a person object of DN <cn=person, o=organization>. Application A wants total and complete control over some subset of the attributes as does Application B. Each application will create a child node with the relative distinguished name (RDN) of the application name: <appName=applicationName, cn=person, o=organization>. This new node

will be of objectclass <applicationNamePersonNode>. The application will store its secret application data in this subnode. As much common information should be shared as possible, but, in cases where it is not feasible, another object is created. One can still effectively retrieve all the attributes of the object by performing a one-level search, which would return the application-specific attributes.

This approach has several benefits. It ensures that the <cn=person, o=organization> object cannot be deleted by anyone who does not have permission to delete all to the application-specific nodes under that object. Therefore, application A does not have to worry about nasty side effects of application B deleting the object. If application B wants the object gone, it deletes its own subNode <appName=applicationB, cn=person, o=organization>, and the entry <cn=person, o=organization> is gone from application B's perspective. However, for application A, the entry still exists.

Using the access control list on the application-specific subnode, the administrator can completely control who has permissions to those attributes. It may be that some are still world-readable where in some cases some other applications find them useful. It is, of course, possible that each application can give delete or modify permission to administrators of one or more other applications. For instance, the user administrator might give modify permissions on some classes to the help desk administrator so passwords or other information can be reset when the user calls the help desk.

4.3 UTF-8 Support

The LDAP V3 protocol defines that string values use UTF-8 (see Figure 20 below). As the name implies, UTF-8 (UCS Transformation Format) is a transformation of Unicode characters into a byte representation that takes up from one to eight bytes per character. As such, UTF-8 gives a clearly defined conversion to and from Unicode while, on the other hand, maintaining compatibility with most existing character encoding, such as ASCII.

Passing data in UTF-8 simplifies and standardizes the handling of national language characters in the protocol and on the server. The client libraries that ship with the IBM SecureWay Directory Client SDK are capable of doing the conversions to and from UTF-8 on behalf of the application.

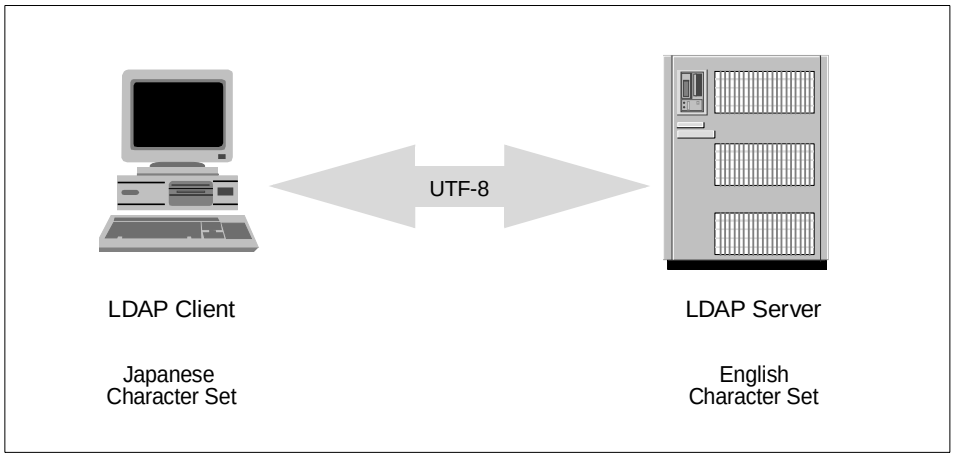


Figure 20. Data passes in UTF-8 character set

By default, the directory server's database (DB2 in the IBM SecureWay Directory) is created using a code set based on the server machine's locale. The directory data is then translated from UTF-8 to the database's code set before it is stored. This limits the support of that directory for other character sets and collation sequences, but, otherwise, does not cause any problems if all clients use the same character set.

The IBM SecureWay Directory supports another option, which is to create the database using the UTF-8 code set. The server will then not need to do code set translation at the DB2 interface, and this makes the directory support multiple character sets. This should be considered when multiple languages are to be supported by the directory. Users should be made aware that, due to the differences in collation sequences of different languages, searches with search filters using *greater than* or *smaller than* operations may not give expected results.

When an application uses LDAP V3, string data is expected to flow on the wire between the client and the server in the UTF-8 character set (Figure 20 on page 84). Most often, the applications operate in the system's local code page (that is not UTF-8). The current client implementation makes the following assumption:

1. String data supplied by the application (for example DNs) is already encoded as UTF-8. If the application is restricting itself to printable ASCII range, the strings are already resident in a proper subset of UTF-8. If the application is using a code set where characters do not fall into a proper

subset of UTF-8, the application must convert the strings from the local code set into UTF-8 prior to sending it to the server.

2. String data returned to the application is returned *as received* from the server, which for LDAP V3 is UTF-8. It is the application's responsibility to perform any necessary conversion to a local code set if required.

The C language API supports separate translation routines for translating strings to and from UTF-8. The JNDI also supports the necessary conversion between Unicode (the native Java code set) and UTF-8. More information about the APIs can be found in Chapter 8, "Developing Directory-Enabled Applications" on page 223.

4.4 Tivoli TME Considerations

Another important consideration for enterprise-wide systems administration may, as a component of an overall management framework, include administering entries in an LDAP directory server. As introduced in 1.7.4, "Tivoli User Administration: LDAP Connection" on page 26, the IBM SecureWay Directory can be integrated with the Tivoli systems management framework using the Tivoli LDAP Connection.

The LDAP Connection is a profile endpoint to aid in the management of user data. The objectives of this service are to distribute and query user data to and from LDAP services. Consider it a two-way bridge between Tivoli user profiles and LDAP directory entries.

The Tivoli Management Environment (TME) release 3.6 adds support for LDAP directory service management. Using the Tivoli User Administration user profiles, entries in an LDAP directory can be managed. Two directions of data flow are supported. The first, a push mechanism, distributes user profile data from the Tivoli TME to an LDAP directory server. The other method populates Tivoli user profiles by pulling data from an existing LDAP directory. It is also possible to schedule synchronization of user data between Tivoli and the LDAP directory.

In the Tivoli database, user profiles exist that contain the user information, such as name, employee number, e-mail address, and so on. Tivoli attributes are mapped to LDAP attributes, which can be configured using the Tivoli

LDAP Attribute Name Mapping function. Table 3 below provides an example of attribute mapping.

Table 3. Tivoli LDAP connection attribute mapping

Tivoli Attribute Name	inetOrgPerson Object Class Attribute Name
real_name	cn
sso_password	userpassword
sso_login	uid

The Tivoli profile manager(s) keeps track of different user profiles and other subscriber services, such as the LDAP endpoint. These user profiles can be used to distribute user information to different endpoints, such as an LDAP directory.

The LDAP Connection must be installed and configured in a Tivoli Management Region (TMR) on any managed node in order to provide the services needed to distribute and query an LDAP directory. The Tivoli LDAP Connection uses the IBM SecureWay Directory Client SDK that supports the LDAP version 3 protocol. The IBM SecureWay Directory server may be installed on any machine to which the client running the Tivoli LDAP Connection has access using LDAP.

As part of the Tivoli administrator authorization scheme, Tivoli administrators can delegate management of selected nodes and endpoints including the LDAP service. This layered administration model is useful when providing access control to data in different parts of the directory hierarchy. Currently, there is no support for ACL management of directory services through the Tivoli LDAP Connection. SSL support can be configured to encrypt data sent over the network.

Before LDAP resources can be managed, the Tivoli LDAP Connection must be added as managed resources to the TMR. When adding an LDAP Connection, you must associate it with an LDAP server.

It may be necessary to install and create multiple LDAP Connections if the directory tree is partitioned.

The installation requirements for the LDAP Connection services are the following:

- Tivoli Framework Version 3.6
- Tivoli User Administration
- Tivoli LDAP Connection

- IBM SecureWay Directory Client SDK

In order for the LDAP service to exchange information with the Tivoli User Administration, some connection properties must be configured. Class and attribute types must be defined in the LDAP Connection from the appropriate LDAP directory object classes and attribute types. This definition will reflect the scope of the management Tivoli performs on the LDAP directory.

The following properties must be known to configure the LDAP Connection:

- Hostname of the LDAP server
- Port number where the LDAP service is available to access
- Distinguished name (DN) that the Tivoli service will use to bind to the LDAP directory
- Password of the above DN identity
- The entry (base DN) in the directory where the Tivoli LDAP Connection should begin its searches
- The class of the objects to be managed in this directory

The level of system administration is limited to the population and distribution of attributes that are supported in the current Tivoli User Administration product. It is important to configure simple LDAP namespaces and topologies, otherwise, the need for several instances of the LDAP Connection may become exhaustively complicated.

4.5 Distributed Directories - Split Namespaces

Following are some considerations to take into account for large directories or when a directory service needs to be available at multiple locations possibly involving local administration authority without reliable network connections in between. There are, generally, two ways to distribute a directory:

- Split Namespaces – Each server only stores a subset (partition) of the whole namespace. All servers together build up the entire namespace.
- Replication – Each server stores the whole namespace. One server is a master while all others are exact replicas of the master.

The major reason for replication is load balancing and availability. See 7.6, “Replication” on page 178 for more details. The following discussion focuses on split namespaces.

4.5.1 Partitioning a Directory

Partitioning a directory tree and distributing it to multiple LDAP servers at multiple locations has many benefits including:

Scalability – More data can be accommodated by the directory since the tree information is stored on a collection of servers, not just a single one. This provides for a (theoretically) indefinite size of namespace.

Manageability – Each location can manage their own part of the directory tree on the local machine. Alternatively, management can also be done centrally.

Availability – Spreading the directory information into subtrees reduces the possibility of a single point of failure. However, one drawback to this approach is that the probability of failure might increase as more systems are involved and depending on how the directory information is accessed. If requests are primarily being handled (and eventually forwarded to other servers) by a single server, the service still depends on a single machine (unless other provisions are in place).

Load Balancing – The work load of the actual data retrieval can be spread among the servers.

A technique to partition a directory tree is to use LDAP *referrals*. LDAP referrals point to a different partition of a namespace stored on a different (or the same) server. For example, if your main directory server is located in New York, and you want to redirect all the requests for <ou=Austin, o=Your_ORG, c=US> to a directory server located in Austin, you can specify this with a referral entry in the main directory tree in the following format:

```
ldap://<hostname:port>/ou=Austin,o=Your_ORG,c=US
```

A referral is a pointer to another portion (partition) of a directory. It is returned by the server to a client and it is then up to the client to follow such a referral.

The steps for partitioning the namespace using referrals are:

1. Plan your namespace hierarchy. For example:

```
country: US
company: IBM, Lotus
organizationUnit: IBM Austin, IBM Raleigh, IBM HQ
```

2. Set up multiple IBM SecureWay Directory servers. Each server contains a part (partition) of the namespace. For example, we assume the following setup for the directory servers:

- The HQ server – A server which contains information about headquarters and contains pointers to other branches within IBM
- The Austin Server – A server which contains information about the Austin site
- The Lotus Server – A server which contains information about Lotus Corporation
- The Raleigh Server – A server which contains information about the Raleigh site

3. Set up referral objects to point to other servers in the hierarchy. You can set up pointers pointing back to the parent or directly to other subtrees in the hierarchy. Let's assume, in our example, that there will be much interaction between the Austin site and Lotus. In this case, it would make sense to set up direct pointers between these two servers rather than only pointing back to the top server of the hierarchy. Figure 21 illustrates the setup.

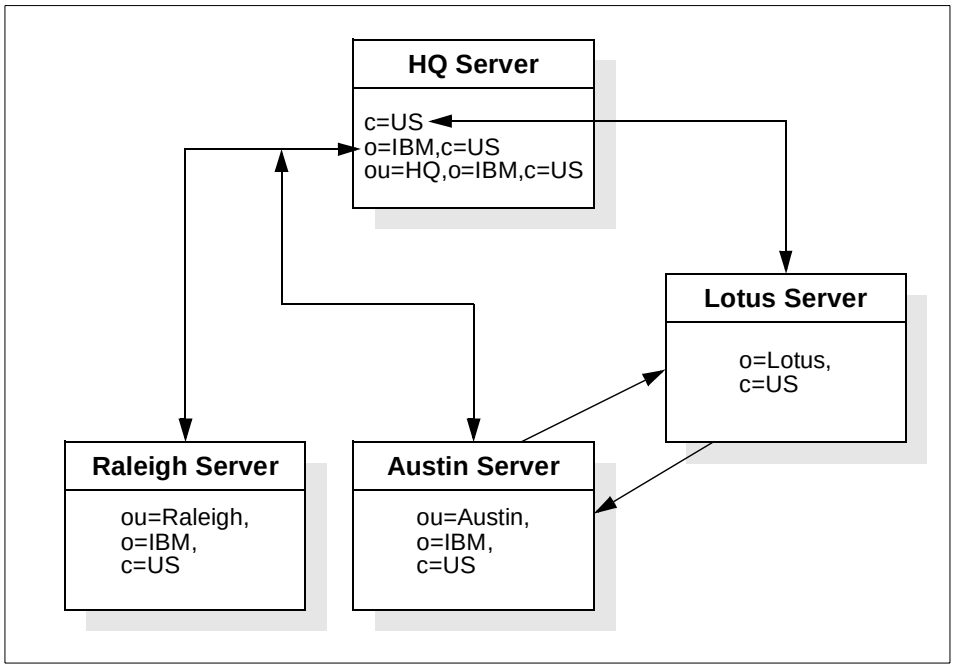


Figure 21. Referrals example

In this setup, requests for information can be done in one of the following ways:

- If a user is searching information related to a specific site, he/she can go to the site server directly and specify the site DN as the base DN. For example, if an IBM Raleigh employee tries to find the telephone number of his team member, he can use the Raleigh server and <ou=Raleigh, o=IBM, c=us> as the base DN for that search.
- If the user is not sure where the information is located, he/she can start the search from the root of the directory tree (base DN: <c=US>), using either the root directory server or the server which is located closest to him. Either way, the search will eventually be pointing at the correct server, provided that the referral pointers are correctly set up. For example, if an IBM Austin employee is trying to find the phone number of an IBM employee outside Austin using <c=US> as the base DN and using the root directory server, the search request will be referred to the server that stores the data through the referral pointer that is returned to the LDAP client. The LDAP client will then follow the referral.
- If a user sends a request to the local directory server to search for information not located on the local server, the local server will return the request back to the client with a referral pointer to either the correct server (if such a direct pointer was defined) or to the root server. For example, if a user at the Austin site is requesting information located at the Lotus server, the Austin server returns a referral pointing to the Lotus server. This is normally transparent to the user, and he/she will not be aware of the fact that the information is actually stored on a different server.

4.5.2 Administering a Split Namespace

In the split namespace environment, each location can optionally manage its own part of the directory tree. This way, each location will need to have a directory administrator defined in its local directory partition. No single user will be able to manage the whole directory in such an environment with a single user credential. Here is why: LDAP referrals automatically forward user requests along with the user's logon credentials (DN and password) via the client to another server. For unauthenticated access, the DN and password will be null, and another server will treat the forwarded request as an unauthenticated request and proceed accordingly.

For authenticated access (for example, for directory administration functions), since the administrator's DN can be defined only on the server which manages the suffix of the DN, the next server in the referral chain will not be able to recognize that user. The user has to rebind to the next server with a different user ID (that is, a different DN).

This has an impact on all LDAP requests that administer data (add/modify/delete) in a split namespace directory implementation. If you are writing an application program which traverses the directory tree through referrals, you should use the ldap_set_rebind_proc() call to set the entry-point of a callback routine (see the on-line *LDAP Programming Reference* that ships with the product for more details). This routine will obtain the bind credentials for use when a new server is contacted during the following of a LDAP referral.

In the example mentioned previously (Figure 21 on page 89), in order for a system administrator to manage the directory entries stored on the Lotus server, he/she has to log on as the system admin for the Lotus server. If the system administrator wants to manage the server at Raleigh, he/she needs to logon as the system admin for the Raleigh server (that is, as a different user). If the system administrator would like to manage all directories under a single ID, putting all directory information on a centralized server is the recommended solution. Alternatively, the administration tool this administrator uses must use the ldap_set_rebind_proc() API call to obtain the correct bind credentials for each partition of the directory.

4.6 Migration from the Previous Release

The IBM SecureWay Directory V3.1 schema file has a different syntax than its predecessor, the IBM eNetwork LDAP Directory V2.1. This was introduced because the IBM SecureWay Directory V3.1 has a richer set of information stored in this file and uses different keywords.

The migration to IBM SecureWay Directory V3.1 involves directory contents (data) migration as well as configuration and schema file migration. IBM provides a migration tool to migrate the configuration and schema file(s) while the IBM SecureWay Directory V3.1 can access and use the data stored in the DB2 database without any further migration steps necessary.

The migration tool is integrated with the installation procedure for the IBM SecureWay Directory V3.1, and it is run transparently to the user (Figure 22 on page 92). Alternatively, it can be run manually from the command line.

The command syntax for the schema migration tool is as follows (make sure the IBM SecureWay Directory server is stopped while running this conversion tool with the active configuration):

```
java migrate <source_dir> <dest_dir>
```

source_dir This is the directory path that contains slapd.conf and V2.1 schema files.

dest_dir This is the directory path that will contain the new slapd.conf and V3.1 schema files.

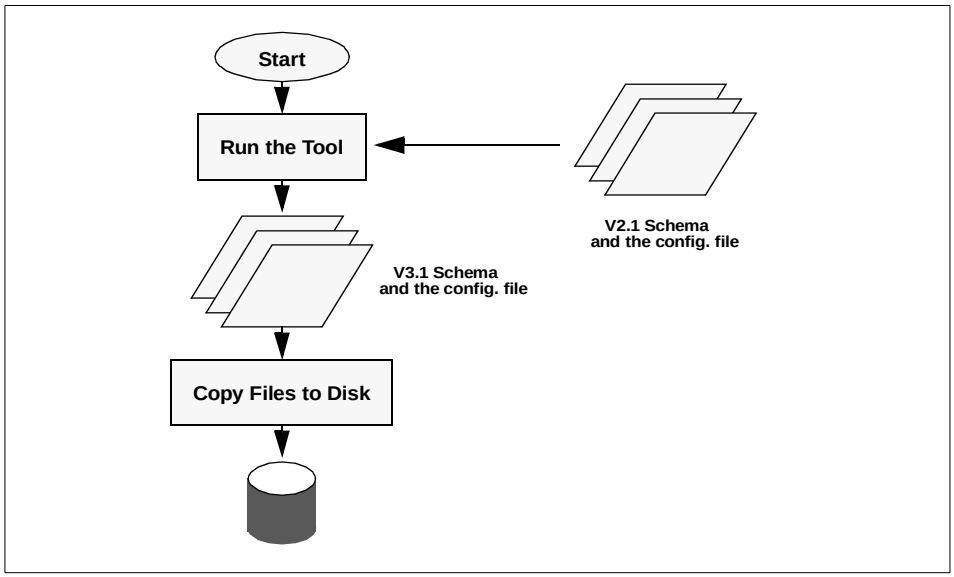


Figure 22. The schema migration tool

As can be seen in Figure 22, the migration process follows the basic steps of:

1. Load the data from the source directory (source_dir).
2. Convert the slapd.conf and V2.1 schema file(s) to the new slapd.conf and V3.1 schema file(s).
3. Install the new files into the target directory (dest_dir).

As mentioned above, the IBM SecureWay Directory V3.1 continues to use the existing database from the eNetwork LDAP Directory V2.1.

4.7 Migration from Non-LDAP Sources

Many of the LDAP directory exploitation projects may involve the re-engineering of existing applications and systems to take advantage of a centralized directory service. One aspect of re-engineering is moving data from an existing side file, directory, or other structure into the IBM SecureWay Directory. This section discusses several approaches to data migration.

Data migration is the process of moving data from one repository to another (in this case, to an LDAP directory). Data migration is part of setting up and maintaining a directory. When creating the directory, the data must be migrated into the directory from another directory, a legacy database, or some more abstract source. Once a directory has been populated with data, you may have occasions to migrate data from one directory server to another. This might occur when replacing a directory server with a newer or different server. Or, it might relate to moving some portion of the directory namespace from one server to another to balance the load.

This section does not cover the replication features that can be set up so that updates to a master server are automatically propagated to a number of slave servers to keep their contents in sync (see 7.6, “Replication” on page 178 for a discussion of this topic).

The currently prevalent approach to importing or exporting data to or from an LDAP directory uses the LDAP Data Interchange Format (LDIF). This format is defined in an Internet Draft (draft-good-ldap-ldif-03.txt, see Appendix A, “Standards” on page 269). Most LDAP directories provide utilities to import data from an LDIF file or to export data to an LDIF file. LDIF is a text-based representation of the data; so, it is relatively easy to work with or to generate from other forms of data. In addition to migration of data, export and import to or from LDIF may be used as a form of backup/restore of directory data.

The most straightforward data migration operation is moving data between two LDAP directory servers from the same vendor. Due to the lack of an IETF-accepted LDAP Access Control model and numerous differences in schema definitions across the industry, moving data from one vendor’s LDAP directory to another may present some additional challenges. Schema definitions may have to be updated. LDIF files may have to be passed through a filter to remove portions of the data that do not map to the target directory.

Because LDIF files are an important tool for directory administrators, it is briefly overviewed in the following sections, and specifics relative to migration are pointed out. For a more detailed description, we refer you to the applicable Internet Draft (see above).

4.7.1 The LDIF File Format

The LDIF format is used to convey directory information or a description of a set of changes to directory entries. An LDIF file consists of a series of records separated by line separators. A record consists of a sequence of lines describing a directory entry or a sequence of lines describing a set of

changes to a single directory entry. An LDIF file specifies a set of directory entries or a set of changes to be applied to directory entries but not both.

The basic form of a directory entry represented in LDIF is:

```
dn: <distinguished name>
objectClass: <object class>
objectClass: <object class>
...
<attribute type>:<attribute value>
<attribute type>:<attribute value>
...
```

The DN and at least one object class definition are required. In addition, any attributes required by the object classes for that entry must also be defined in the entry. All other attributes and object classes are optional. You can specify object classes and attributes in any order. The space character after the colon is optional.

Table 4 below describes the LDIF fields shown in the previous definition of a directory entry in an LDIF file.

Table 4. LDIF fields

Field	Definition
dn: <distinguished name>	Specifies the distinguished name for the entry.
objectclass: <object class>	Specifies an object class for the entry. Object classes determine the types of attributes that are required for the entry as well as the attributes allowed for the entry.
<attribute type>	A name identifying an attribute type. Attribute types are defined in the schema of the LDAP server and determine the types of values that are allowed.
<attribute value>	A value for the attribute. The <attribute type> and <attribute value> together make up an Attribute Value Assertion (AVA) that specifies a particular value associated with an entry.

For a more complete description of the LDIF file format, we refer you to the Internet Draft mentioned in the introduction of this section (or see Appendix A, “Standards” on page 269).

4.7.2 LDIF Data Encoding

Any line in an LDIF file may be wrapped by inserting a line separator and a space. Any line which begins with a single space is treated as a continuation of the previous line.

Any line which begins with a pound-sign (“#”, ASCII 35) is considered a comment line and is ignored when parsing the file.

Binary data, such as a JPEG image, can be represented in LDIF by using Base64 encoding. Base64 encoded data is identified by using the double-colon (::) symbol as in the following example:

```
jpegPhoto:: <encoded data>
```

In addition to binary data, other values that must be Base64 encoded include:

- Any value that begins with a colon (“:”, ASCII 58), less-than (“<”, ASCII 60), or a space (“ “, ASCII 32).
- Any value that contains data outside of ASCII values 32 - 255 decimal.

4.7.3 Creating Directory Entries Using LDIF

There are many types of entries that can be stored in a directory. This section will show three of the most common types of entries used in a directory: *organization*, *organizational unit*, and *organizational person* entries.

The object classes defined for an entry are what indicates whether the entry represents an organization, an organizational unit, an organizational person, or something else entirely different from these types of entries.

4.7.3.1 Specifying Organization Entries

Most directories have at least one organization entry. Typically, this is the first, or root, or topmost entry in the directory. The organization entry often corresponds to the suffix set for the directory. That is, if the directory is defined to use a suffix of <o=ibm.com>, the organization will probably have an entry in the directory named <o=ibm.com>. The LDIF entry that is specified to define an organization entry should appear as follows:

```
dn: <distinguished name>
objectClass: top
objectClass: organization
o: <organization name>
<list of optional attributes>
...
```

The following is a sample organization entry in LDIF format:

```
dn: o=ibm.com
objectclass: top
objectclass: organization
o: ibm.com
telephonenumber: 123-4567
```

4.7.3.2 Specifying Organizational Unit Entries

There is usually more than one organizational unit or branch point within a directory tree. The LDIF that you specify to define an organizational unit entry should appear as follows:

```
dn: <distinguished name>
objectClass: top
objectClass: organizationalUnit
ou: <organizational unit name>
<list of optional attributes>
...
```

The following is an example organizational unit entry in LDIF format:

```
dn: ou=people, o=ibm.com
objectclass: top
objectclass: organizationalUnit
ou: people
```

4.7.3.3 Specifying Organizational Person Entries

The most common type of entry that will be included in directories will describe a person within the organization. The majority of the entries in the directory will represent organizational people. The LDIF used to define an organizational person should appear as follows:

```
dn: <distinguished name>
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: <common name>
sn: <surname>
<list of optional attributes>
...
```

The following is an example organizational person entry in LDIF format:

```
dn: cn=John Smith, ou=people, o=ibm.com
objectclass: top
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: John Smith
```

```
sn: Smith
givenname: John
uid: jsmith
ou: Marketing
ou: people
telephonenumber: 838-6004
```

4.7.4 LDIF File Example

Following is a simple LDIF file which contains two organizational units beneath the organization ibm.com. The entry of John Smith is the only data entry for the *people* organizational unit.

The order of entries is important. An entry can only be added to a directory if it is a *suffix* or *naming context* entry (that is, a root of a local directory tree), or its parent entry has already been added. So, in the example below, the first entry will normally be configured as a suffix for the server, and the subsequent entries all come after their parent entry.

```
dn: o=ibm.com
objectclass: top
objectclass: organization
o: ibm.com
dn: ou=People, o=ibm.com
objectclass: organizationalUnit
ou: people
dn: ou=marketing, o=ibm.com
objectclass: organisationalunit
ou: marketing
dn: cn=John Smith, ou=people, o=ibm.com
objectclass: top
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: John Smith
sn: Smith
givenname: John
uid: jsmith
ou: people
telephonenumber: 123-4567
```

4.7.5 Importing LDIF Data

If the directory you are importing the LDIF data to already contains entries that match some of those in your LDIF file, you will get *entry already exists* errors on those entries. In this case, look for an option of the import utility that will continue importing entries even if errors occur. The import utility should also give some indication of the number of entries successfully added to the

directory. This allows you to easily verify whether all entries have successfully been added.

4.7.5.1 Moving Data Between Homogeneous Directories

When moving data between directories, there are several factors that influence the complexity of the task. From an LDAP perspective, these factors include the hardware and software platform of the server, the vendor supplying the directory software, and the implementation or support of LDIF. Beginning with the simplest, these factors present the following interesting combinations of source and target directory:

- Homogeneous: same vendor, product, and version (same or different hardware platform)
- Heterogeneous: different vendor, product, and version (both LDAP, supporting LDIF)
- Legacy data: from non-LDAP directory or database

The differences and difficulties involved in moving data between platforms tend to be minor compared to those introduced when moving data between directories from different vendors.

When homogeneous directories are involved, the process of moving data from one server to another follows the basic steps of:

1. Export the data from the source directory server to an LDIF file.
2. Transport the LDIF file to the target directory server system.
3. Import the data from the LDIF file into the target directory.

Before beginning this data migration process, compare the schema of the source and target directories. If the source directory server has definitions of attribute types or object classes that are not included in the target directory's schema, you should update its schema to add them to ensure that the target directory can accept all the data exported from the source.

Note

The method used to update the schema in the target directory is vendor, product, and release dependent (for example, the IBM eNetwork LDAP Directory V2.1 and the newer IBM SecureWay Directory V3.1 do not use the same schema file syntax).

The IBM SecureWay Directory server includes the command line utilities db2ldif to export data to an LDIF file and ldif2db to import from an LDIF file.

The Web browser-based administration GUI (UNIX and Windows NT platforms only, see 7.2, “The Administrator Graphical User Interface” on page 166) also provides these export/import interfaces. When moving data between two instances of the directory server, the steps might look like:

1. Export the data on system A:

db2ldif -o dirdata.ldif

2. Move the LDIF (text, not binary) file to system B, for example, using FTP.
3. Import the data on system B:

ldif2db -i dirdata.ldif

Note

Data exported in LDIF format will be in readable format. Entries in the directory that were protected by controlling access will be exported without regard to access control rights.

4.7.5.2 Moving Data Between Heterogeneous Directories

The process of transferring data between LDAP directories can be more difficult if the two directories are from different vendors. This is due to differences in access control models, which have not yet been standardized for LDAP, as well as other features that may or may not be identical.

As an example of the sort of problems that might be encountered, let’s take a closer look at moving data from a Netscape Directory Server to an IBM SecureWay Directory Server. The definition of access control is quite different between the two directories. It is different enough that it would not be practical to attempt to transfer the access controls along with the data. Instead, you will need to strip out the access control information from the LDIF file, load the data into the IBM SecureWay Directory, and then use the administrative interfaces to redefine the appropriate access controls.

When the data is dumped to an LDIF file from the Netscape directory, the access controls (if any were defined) will appear as *ACI* attributes with the entries. These lines of the file should be removed prior to loading the data to the IBM SecureWay Directory directory. For example, on a UNIX platform, these lines can be very easily removed using the following command:

```
# grep -v “^ACI” filename.ldif > newname.ldif
```

Examples of ACL information in LDIF form for each of the directory implementations follow.

Example IBM SecureWay Directory ACL format:

```
dn: cn=John Doe, ou=Austin, o=IBM, c=US
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: John Doe
sn: Doe
dept: abcd
aclEntry: access-id:cn=Bert Hello, ou=Austin, o=IBM,
 c=US:object:da:normal:rwsc:sensitive:rsc
 aclEntry: group:cn=Anybody::normal:rsc
 aclEntry: access-id:cn=Amy Too, ou=Austin, o=IBM,
 c=US:object:da:normal:rwsc:sensitive:rsc:critical:rsc
aclPropagate:TRUE
entryOwner: group:cn=personnel, ou=Austin, o=IBM, c=US
ownerPropagate: TRUE
inheritOnCreate: FALSE
```

Note

- If you do not want an access ID to have permission to a particular class, do not specify the class.
- Attributes are assigned to their respective classes in the slapd.at.conf file.

Example Netscape Directory representation of Access Controls:

```
dn: o=airius.com
objectClass: top
objectClass: organization
ACI: (target = "ldap:///cn=administrators,o=airius.com")
 (targetattr="member||uniquemember")(version 3.0; acl
 "write-admingrp"; allow (write)userdn = "ldap:///uid=ssarette,
 o=airius.com" ||
 "ldap:///uid=bjensen, o=airius.com";)
```

As you can see from these examples, the access control policies and representations are very different. The approach of removing and then reapplying access control will be similar when moving data from an IBM SecureWay Directory to a Netscape directory. In this case, the attributes to be removed from the (IBM SecureWay Directory) LDIF file include *aclEntry*, *aclPropagate*, *entryOwner*, and *ownerPropagate*.

In addition to differences in access control, there may be other differences in the attributes that are valid from one server to another. Compare the schema definitions of the servers you are using. In some cases, the schema for the

target directory can be modified to add attribute definitions from the schema of the originating directory server. In other cases, there may be attributes related to extended features that are not part of the LDAP standard and, therefore, cannot be added to the target server. In those cases, it may make more sense to strip those particular attributes out of the LDIF file as was done for the ACI attributes.

4.7.5.3 Migrating Data to LDAP from Legacy Directories/Databases

Lastly, if you have data in legacy databases or directories (not based on LDAP), you may well wish to migrate your data to an LDAP directory. The use of a well-defined, text-based interchange format facilitates development of tools to import data from legacy systems. A fairly simple set of tools, for example, written in UNIX shell script language, can convert a database of personnel information into an LDIF file, which can then, in turn, be imported into the LDAP directory regardless of the internal database representation the target directory server uses.

4.8 Summary and Conclusions

There are multiple ways to administer data in the IBM SecureWay Directory and in the server itself. It is important to understand the options provided including APIs, utilities, and GUI-based tools. It is also imperative that these functions are implemented with a clear understanding of who, by role or responsibility, should perform them.

Any customer may implement the IBM SecureWay Directory and modify the default namespace, schema, policy, and so on. The recommended approach for any directory projects should be to assign much of the responsibility to the directory administrator. The role for Tivoli LDAP Connection is limited to person and group entries.

The administrative capabilities (to add, change, or delete directory data entries, schema definition, and access control) of LDAP-enabled applications must be limited to those elements that are not shared with other applications. The following table summarizes the administrative responsibilities of the directory administrator, LDAP enabled applications, and the Tivoli LDAP Connection.

Table 5. Administration matrix

Function	Directory Administrator	LDAP Enabled Application	Tivoli TME LDAP Connection
Query Entries	OK	OK	OK

Function	Directory Administrator	LDAP Enabled Application	Tivoli TME LDAP Connection
Add/Modify/Delete Non-Shared Entries	OK	OK [1]	-
Add/Modify/Delete Shared Entries	OK	-	OK [2]
Add/Modify/Delete Schema	OK	OK [1]	-
Back-Up/Restore Entries	OK	-	-
Back-Up/Restore Schema	OK	-	-
Define Access Policy	OK	-	-
Change Access Policy	OK	OK [1]	-

[1] Data, Schema, Access Control for non-shared elements only
[2] Users and groups only

Chapter 5. Directory Security

Security is very important in the networked world of computers, and this is equally true for directories as well. Directories are likely to contain sensitive information that needs to be protected from unauthorized access and modification. When sending data over networks internally or externally, sensitive information may also need to be protected against eavesdropping and modification during transportation. There is a need to know who is requesting the information and who is sending it.

After a brief introduction to the subject, this chapter explains the security features supported by the IBM SecureWay Directory and how to configure them.

5.1 Security of the Directory

The directory may contain many types of information ranging from publicly accessible data, such as e-mail addresses, to very sensitive data like user passwords. Hence the appropriate level of security for the data in the directory must exist.

In particular, the IBM SecureWay Directory provides (explained in the section that follows):

Authentication The requester must prove his/her/its identity to the directory. This is supported using the SASL/CRAM-MD5 mechanism and certificates using SASL/SSL.

Access Control The directory server only returns data that the requester is entitled to access. In other words, the requester must have adequate *authorization*. This is implemented through the use of Access Control Lists (ACLs).

Integrity Data needs to be reliably stored and transmitted such that alterations can be detected. SSL network transmissions are protected against alterations.

Confidentiality (Also called *privacy*) Sensitive data transmitted to/from the directory or stored in the directory cannot be easily accessed without proper permission (authorization). User passwords can be stored encrypted in the directory. Network transmissions can be protected using SSL.

5.2 Security Support of the IBM SecureWay Directory

The IBM SecureWay Directory supports the following methods for authentication and privacy:

- Anonymous authentication
- Basic authentication (distinguished name and password)
- Simple Authentication and Security Layer (SASL)
- Secure Sockets Layer (SSL) certificate authentication and encryption

In an LDAPv3 implementation, the client must authenticate itself to the directory service before accessing any data in the directory, otherwise an error is returned.

Anonymous authentication is useful for read-only access of directory data where that data is not sensitive, such as peoples' e-mail addresses or office numbers. Essentially, that data is accessible to *anyone*. To request anonymous authentication, simple authentication is performed with a distinguished name (DN) that is empty.

Basic authentication provides authentication facilities with the DN and password transmitted over the network in clear text. Use of clear text passwords is not recommended over open networks when there is no authentication or encryption being performed by a lower layer, such as SSL (see below). Access (read or write) to directory data is granted based on DNs contained in the access control list of the object and/or attributes in the access request.

The Simple Authentication and Security Layer (SASL) is a framework for multiple authentication and encryption mechanisms for connection-oriented protocols, as described in *Simple Authentication and Security Layer (SASL)*, RFC 2222. It has been added to LDAP Version 3 to overcome the authentication shortcomings of LDAP Version 2. The following section, 5.2.1, "Overview of Simple Authentication and Security Layer (SASL)" on page 105, explains SASL in more detail.

The Secure Sockets Layer (SSL*)* is, strictly speaking, not part of LDAP. It is a lower-level authentication and encryption service that higher-level applications, such as LDAP or Web browsers, can use and rely on. It provides strong authentication using certificates and strong encryption using the Data Encryption Standard (DES) for securing the network traffic (encryption levels are subject to country regulations). SSL is widely used in the industry.

Section 5.2.2, “Overview of Secure Sockets Layer (SSL)” on page 106 and the sections that follow explain SSL and its configuration.

5.2.1 Overview of Simple Authentication and Security Layer (SASL)

In SASL, connection protocols, such as 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. 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. LDAP Version 3 includes such a command: `ldap_sasl_bind()`. Optionally, a security layer can be negotiated to encrypt the data after authentication and thus ensure confidentiality. The IBM SecureWay Directory supports SASL authentication using the Challenge Response Authentication Mechanism with Message Digest 5 (CRAM-MD5) mechanism, which transmits message digests rather than the passwords themselves over the network.

The key parameters that influence the security method used are:

- DN – This is the distinguished name of the entry a requester wants 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. The IBM SecureWay Directory supports *CRAM-MD5* and *external*. There is also an *anonymous* mechanism available which enables an authentication as the generic user *anonymous*. In LDAP, the most common mechanism used is SSL (or its successor TLS), which is provided as a so-called *external* mechanism.
- Credentials – This contains the arbitrary data that identifies the DN. The format and content of the parameter depend 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 (sometimes also referred to as *certificate bind*), 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 an intermediator between the authentication system and a protocol like LDAP.

There is no special configuration necessary on either side (client or server) to use SASL/CDRAM-MD5 authentication. Applications simply request it by calling the appropriate API call. The SASL bind operation is explained in

more detail with an example in 8.1.4, “JNDI and Security” on page 230. If CRAM-MD5 authentication is being used, user passwords cannot be stored encrypted because the algorithm must be able to retrieve them in the clear (see also 5.7.1.3, “Password Encryption” on page 128).

If you want to verify which SASL mechanisms an LDAP server (Version 3 only) supports, point a Web browser to a Web site like this:

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

This is an LDAP URL, very similar to those used for HTTP (`http://<host>/...`) or other Internet protocols, as introduced in 2.2.2, “URL Form” on page 34.

As stated earlier, SSL (and its successor TLS) is the external mechanism commonly used in SASL by LDAP for data encryption. Following is a brief description of SSL.

5.2.2 Overview of Secure Sockets Layer (SSL)

The IBM SecureWay Directory Server provides the ability to protect both client and server LDAP access with Secure Sockets Layer (SSL) security. When using SSL to secure LDAP communications with the IBM SecureWay Directory, two forms of authentication are supported:

- Server authentication
- Client and server authentication

SSL is an industry-standard security protocol that uses symmetric-key and public-key cryptographic technology. Symmetric-key cryptography uses the same key to encrypt and decrypt messages. Public-key cryptography uses a pair of keys: a public key and a private key. Each server's public key is published, and the private key is kept secret. To send a secure message to the server, a client encrypts the message using the server's public key. When the server receives the message, it decrypts the message using its private key. Only the server can encrypt this message because it can only be decrypted with the server’s private key.

SSL was developed by Netscape Communications Corp. 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 provides three basic security services:

- Mutual authentication – Mutual authentication is the process whereby the client and the server convince each other of (and prove) their identities. The client and server identities are encoded in public-key certificates. A public-key certificate contains the following components whereby the *issuer*, also known as a Certificate Authority (CA), is a trusted organization, such as RSA Data Security Inc. or Verisign Inc.:
  - Subject's distinguished name
  - Issuer's distinguished name
  - Subject's public key
  - Issuer's signature
  - Validity period
  - Serial number

  Rather than mutual authentication, which provides for maximum security, many implementations only use server authentication.
- Message privacy – Message privacy is achieved through a combination of public-key and symmetric key encryption. All traffic between an SSL client and an SSL server is encrypted using a key and an encryption algorithm negotiated during session setup.
- Message integrity – The message integrity service ensures that SSL session traffic does not change while en route to its final destination. SSL uses a combination of public/private keys and hash functions to ensure message integrity.

As mentioned earlier, SSL is not actually a part of LDAP, though LDAP does use it if appropriately configured. SSL support for the IBM SecureWay Directory requires a separately installable module (see 5.3, “SSL Utilities” on page 108). If SSL is being used to protect LDAP sessions, the SSL session has to be established before *normal* LDAP protocol conversations can take place. Simplified, the following exchange takes place in order to set up a secure SSL session:

1. The client and the server exchange hello messages to negotiate the encryption algorithm and hashing function (for message integrity) to be used for the SSL session.
2. The client and server exchange X.509 certificates to validate their identities (if client authentication is not requested, only the server sends its certificate). Certificates are verified by checking the correctness of format and validity dates and by verifying that the certificate bears the signature of a trusted Certificate Authority (CA).

3. The client randomly generates a set of keys that are used for encryption. The keys are encrypted using the server's public key and securely communicated to the server.
4. Encrypted communication can now start using the generated key for encryption and decryption.

For *server authentication* to function, the IBM SecureWay Directory server must have a digital certificate (based on the X.509 standard). This digital certificate is used to authenticate the IBM SecureWay Directory server to the client application(s). During the initial SSL handshake, the LDAP server supplies the client with its X.509 certificate. If the client validates the server's certificate, a secure, encrypted communication channel is established between the LDAP server and the client application.

If *client and server authentication* is to be used, both the LDAP server and the client application must have a digital certificate. The server's digital certificate is used to authenticate the LDAP server to the client application (for example, an application built with IBM's LDAP application development toolkit). Similarly, the client's digital certificate is used to authenticate the client to the LDAP server (in terms of SSL's strong authentication mechanism). During the initial SSL handshake, the LDAP server and the client exchange certificates for mutual validation. After the client validates the server's certificate and the server validates the client's certificate, a secure encrypted communication channel is established between the LDAP server and the client application.

The following two sections introduce the IBM SSL utilities and the steps necessary to configure SSL.

5.3 SSL Utilities

The graphical utility ikmgui utility (IBM Key Management GUI) is provided for IBM AIX, Windows NT, and a number of other IBM and non-IBM platforms to manage SSL X.509v3 certificate databases (also known as *keyring files* or *keyring databases*). Its use is required to configure and use Secure Sockets Layer (SSL). The ikmgui utility replaces the mkkf and ikeyman utilities used with earlier versions of IBM SSL support. With the IBM SecureWay Directory (and associated clients), both client and server keyring files are managed with ikmgui.

The ikmgui utility, together with the SSL libraries, form the IBM SSL toolkit known as GSKit (Global Security Kit). GSKit provides the SSL protocol functions as well as a set of Certificate Management Services (CMS) functions. These CMS functions provide access to the certificate database

(the keyring file) as well as functions such as validating client certificates (including Certificate Revocation List processing). The current version is GSKit Version 3, which supports SSL Version 3.0, C/C++ for clients and servers and Java for clients.

GSKit is not available for OS/400. Instead, in order to configure and use SSL on OS/400, you need to install Digital Certificate Manager (DCM), Option 34 of OS/400, and the IBM Cryptographic Access Provider licensed program with program number 5769-AC1, AC2, or AC3. Please read the documentation pertinent to these options for installation and configuration instructions.

Starting with V2R7, the LDAP server on OS/390 uses SystemSSL, which, as part of the Common Data Security Architecture (CDSA), ships with the OS/390 base. It is based on the GSKit as described above for other platforms, such as AIX and Windows NT. You should refer to the respective system documentation or visit the Web site `http://www.ibm.com/s390`.

Since strong encryption (as provided by SSL) is controlled by export and other regulations in the U.S. and other countries, different versions of GSKit exist for different countries. While the installable options differ among these versions, the user interface and configuration steps are generally the same as described in the following sections.

Note on Government Regulation

Encryption technology is subject to government regulations in the U.S. and other countries. Such regulations have changed recently and may change in the future. Due to this, the SSL packaging and implementation may be different as the product rolls out or may change thereafter.

5.3.1 GSKit Installation

GSKit is a separately installable option required only when SSL security is to be used. GSKit might already be installed on your system if another application required it to be installed. GSKit is shipped with the IBM SecureWay Directory in the appropriate version for your country. Please check for and follow any installation instructions that came with the product.

For your convenience, the following are some hints for Windows NT and AIX (for OS/400 and OS/390; see the comments in the last section):

Windows NT – GSKit uses the Windows InstallShield for installation. GSKit normally ships as a self-extracting ZIP file that you need to unpack first. For

example, on a command line, change to the directory where the GSKit file resides and type:

```
gskru301 <path> /d
```

The <path> parameter (optional) specifies a destination directory for the unpacked files and /d (optional) specifies that subdirectories will be created. If no parameters are given, the files are unpacked to the current directory. Note that the filename (gskru301.exe) may be different depending on the actual version you are using. The setup.exe file (one of the unpacked files) for GSKit must be called from the command line with a parameter as follows (do not double-click on the file icon to start it):

```
setup ldap
```

This registers LDAP as an application for GSKit. A second command line parameter allows you to select an installation language other than English. Refer to the documentation for a list of available languages. If GSKit was already installed, you must still run this command, which will then add LDAP as an application (installation will not actually take place again).

AIX – GSKit is shipped as one or more installable fileset(s). Installation should be done using the `installp` command (or `SMIT install_latest`).

After the GSKit is installed, the command `ikmgui` is available to start the Graphical User Interface.

5.3.2 The ikmgui Utility

The `ikmgui` utility with its graphical user interface is used to manage certificates. The specific tasks you can perform with `ikmgui` include:

- Create a key pair and request a certificate from a CA
- Receive a certificate into a keyring file
- Change a keyring password
- Show information about a key
- Delete a key
- Make a key the default key in the keyring file
- Export a key
- Import a key into the keyring file
- Designate a key as a trusted root
- Remove trusted root key designation

To run `ikmgui` on AIX, you need to have the Java Development Toolkit (JDK) installed and the JAVA_HOME environment variable pointing to its root directory. For example, the following commands run in a default installation:

```
# export JAVA_HOME=/usr/jdk_base
# ikmgui
```

After starting, the main window of ikmgui is presented as shown in Figure 23 .

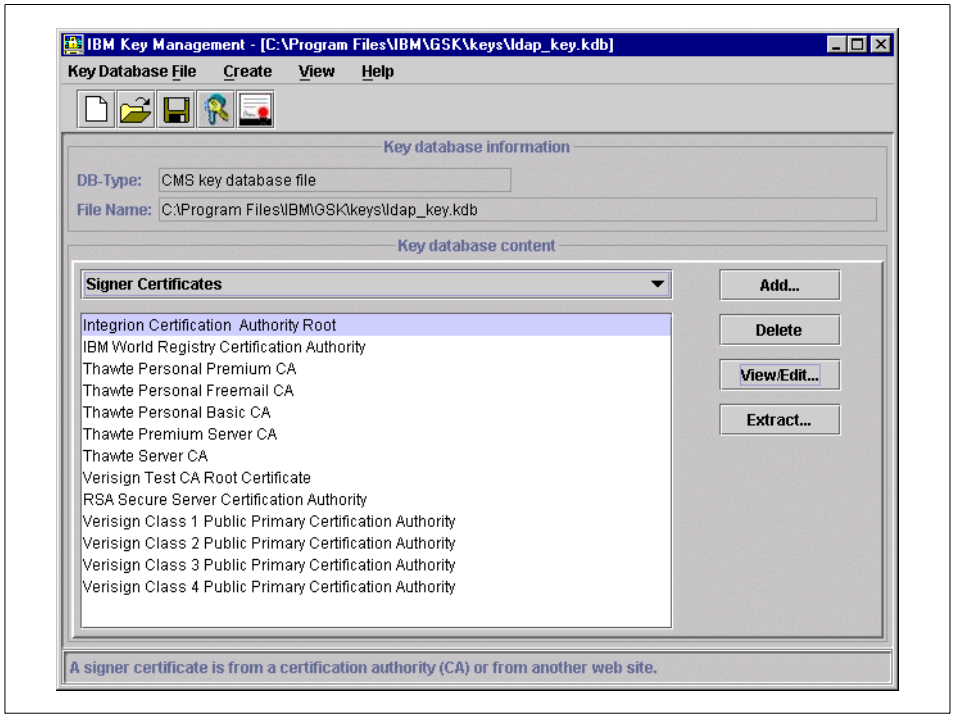


Figure 23. IBM key management (ikmgui) utility main window

The tasks that you need to run SSL from the list above are explained in the following sections.

5.4 Configuring SSL Security

This section describes the procedures required to set up SSL security. To enable security with server authentication, you must do one of the following:

- Create a certificate signed by a well-known certificate authority (CA)

  Create a public/private key pair and obtain and store a certificate from one of the predefined (well-known) Certificate Authorities. This procedure requires less setup because the keyring file is preconfigured with the CA root certificates required to identify the CAs from whom the certificate is issued.

- Create a self-signed certificate

  The process of applying for and receiving a certificate from a CA can take two to three weeks. To enable SSL security until you receive the required CA root and server certificates, you can create a self-signed root certificate and store the certificate in the database and class files. To ensure maximum security for your site, you should only use a self-signed certificate for server authentication until you receive a CA-issued certificate.

Once you have either an official or a self-signed certificate, the server(s) and client(s) can be configured to use SSL. The following sections explain these steps.

5.4.1 Creating a Certificate Signed by a Trusted Certificate Authority

Using a certificate that was signed by a well-known (trusted) certificate authority gives you the advantage that most SSL communication partners know and trust that CA, and they will, therefore, most likely (depending on their configuration) accept a new certificate. This is especially helpful when communicating with partners outside your organization and beyond your authority to change security options. The disadvantages are that it takes some time (a few days or weeks) to get an official certificate and the fact that it is not for free.

Creating a certificate signed by a well-known CA involves the creation of a key database and a certificate request that is then sent to the CA. After returning the certificate from the CA, it needs to be stored in the key database. These steps are detailed below using the GUI of the ikmgui utility.

1. Create server key database (.kdb file):
   - Select **New...** from the on Key Database File pull-down menu on the top of the main window (Figure 23).
   - On the dialog window that pops up, select **CMS key database file** in the Key database type selection list and then type in the name and location of the key database file to be created. This file has an extension of .kdb, as, for example, in ldap_key.kdb. Then, click on **OK** to close the dialog panel.
   - A new dialog pops up that requests your input for a password for the key database file, an optional expiration time, and whether or not the password is to be stashed to a file. Enter a password, an optional expiration time, and make sure that you check the check box next to Stash the password to a file?; otherwise, the IBM SecureWay Directory cannot open the database file. Click on **OK** to close this dialog. The

password is then encrypted and stored in a file with the same name as the key database file but with an extension of .sth.

Your database file is now created. To proceed, you can now create a certificate request.

2. Create a certificate request.
   - Select **New Certificate Request...** from the Create pull-down menu in the main window. In the dialog window that shows up, you will have to fill in the following information for the request:
     - Key label (a clear, descriptive label for the certificate)
     - Key size (512 or 1024, depending on security requirements and country version of the ikmgui utility)
     - Common name
     - Organization and other pertinent information to identify the owner of the certificate
     - Full path of the file name for the certificate request file
   - Click on **OK** to create the request

   The file created (default file name is certreq.arm) contains the certificate request.
3. Send the certificate request, that is, the certreq.arm file, to the certificate authority of your choice by mail or Web (follow their instructions, which can be found on their Web sites).

   (While you are waiting for the certificate authority to process and return your certificate, you can enable SSL security by creating, storing, and importing a self-signed certificate using the procedure described in the next section.)

   Once the certificate has been returned to you by the CA, you have to store it into the key database file.
4. Store the certificate into your database.
   - On the ikmgui main menu (Figure 23), make sure that your key database file is open (check the filename in the Key database information portion of the window). If it is not open, choose **Open...** from the Key Database File pull-down menu and open your file.
   - Select **Personal Certificates** from the selection list in the lower Key database content portion of the window.
   - Click on **Receive...** on the right of the window.
   - Supply the information about the file containing the signed certificate and click on **OK**.

This adds the certificate to the key database file. You will see the new certificate in the list under Personal Certificates.

5. A root certificate of the CA must be stored in the key database file. By default, root certificates of the most common CAs are already present in the file; so, you do not need to add them again. A trusted root is simply an X.509 certificate that has been signed by a trusted entity (for example, Verisign). You can see what root certificates there are by selecting **Signer Certificates** from the selection list in the Key database content portion of the main window. If your CA is not present in that list, obtain a root certificate from this CA and add it by clicking on **Add...** on the right of the window.

This concludes the creation of a certificate signed by a well-known CA. You may choose to created a self-signed certificate instead (or while waiting on the certificate from the CA), which is described in the following section.

5.4.2 Creating a Self-Signed Certificate

You can use the ikmgui utility to create a self-signed certificate to enable SSL sessions between clients and servers. The steps are essentially the same except that, in this case, you are your own CA, and you will be creating your own root certificate. The advantages of using this type of certificate is a quick start, it is free, and you have no dependencies on other organizations. The drawback, on the other hand, is that each client or server using this kind of certificate needs to have the new root certificate imported, which may impose some administrative burden.

Use the following procedures to set up your site with a self-signed certificate, using the ikmgui utility (the first step is identical with the described above).

1. Create server key database (.kdb file).
   - Select **New...** from the on Key Database File pull-down menu on the top of the main window (Figure 23).
   - On the dialog window that pops up, select **CMS key database file** in the Key database type selection list and then type in the name and location of the key database file to be created. This file has an extension of .kdb, as, for example, in ldap_key.kdb. Then, click on **OK** to dismiss the dialog panel.
   - A new dialog pops up that requests your input for a password for the key database file, an optional expiration time, and whether or not the password is to be stashed to a file. Enter a password, an optional expiration time, and make sure that you check the check box next to Stash the password to a file?; otherwise, the IBM SecureWay Directory

cannot open the database file. Click on **OK** to close this dialog. The password is then encrypted and stored in a file with the same name as the key database file but with an extension of .sth.

Your database file is now created. To proceed, you can now create a self-signed certificate.

2. Create a self-signed certificate.
   - Select **New Self-Signed Certificate...** from the Create pull-down menu in the main window (Figure 23). In the dialog window that shows up, you will have to fill in the following information:
     - Key label (a clear, descriptive label for the certificate)
     - Key version (normally *X509 V3*, unless you have reasons for other versions)
     - Key size (512 or 1024, depending on security requirements and country version of the ikmgui utility)
     - Common name
     - Organization and other pertinent information to identify the owner of the certificate
     - Validity period in days
   - Click on **OK** to create the certificate

   This creates a certificate and adds it to the list of Personal Certificates shown in the main window.
3. From the certificate just created above, you need to extract the root certificate that is necessary for other communication partners (clients and/or servers) to recognize the newly created certificate. Here are the steps for exporting the root certificate:
   - Select the new certificate's entry in the Personal Certificate list and click on **Extract Certificate...** on the bottom right in the main window.
   - Select **Base64-encoded ASCII data** from the Data type list and enter a file name (with an .arm extension) and a location (directory) for the new root certificate to be exported to. Then click on **OK** to export the root certificate. (If you want to create a file for the JNDI SSLight client key class, you must select **SSLight key database class** as data type when creating a file with an .class extension. See also 8.1.4, "JNDI and Security" on page 230.)

   You have now created a file that holds your own root certificate. This must be imported to all communication partners that will use SSL to connect to this machine.
4. Use the following steps for importing the new root certificate into others' key database (using ikmgui):

- Make sure the file created in the previous step is available from this system, for example, on a diskette.
- Invoke the ikmgui utility on the receiving system.
- If not already done, create a key database file (see first step above for creating a self-signed certificate).
- In the Key data contents portion of the window, select **Signer Certificates** from the selection list and click on **Add...** on the right.
- Select **Base64-encoded ASCII data** from the Data type list and type the certificate file name and location into the appropriate fields. Then, click on **OK** to import the certificate.
- On the upcoming dialog, supply a label for this certificate and click on **OK**.

The steps as described above need to be done on each machine that will communicate using this certificate with the machine on which the certificate was created.

Each LDAP server should have its own certificate. Sharing certificates across multiple LDAP servers is not recommended. By using different certificates and private keys for each server, your security exposure is minimized should a keyring file for one of the servers be compromised.

5.4.3 Configuring an LDAP Server to Use SSL

After setting up SSL on the lower layers, you have to configure your LDAP Server to use SSL. To do so, follow these steps:

1. On the LDAP Administration Graphic User Interface (see also 7.2, “The Administrator Graphical User Interface” on page 166), click on **Server** and then on **SSL** in the Navigation area.
2. In the working area, select **SSL On** (or **SSL Only**, as appropriate)
3. Choose whether you want Server Authentication or Server and Client Authentication.
4. Set the secure port number (the default is set to 636).
5. Enter the key database file name (including path) and the key label in the appropriate fields.
6. Click on **Apply**.
7. Click on **V3 cipher** (on the left of the working area) and select the desired cipher from the available cipher algorithms (note that multiple selections are allowed) and then click on **Apply**.

8. The server needs to be restarted for the changes to work. This can be done through the Administration GUI.

An easy way to check whether or not an SSL connection can be established is to point a Web browser at `ldaps://<server>/` (note the `s` in `ldaps`) and check the result. The LDAP server should return the contents of the rootDSE (server-specific configuration data), which should be the same as what you get without SSL, that is, by using the non-secure URL: `ldap://<server>/`.

5.4.4 Configuring an LDAP Client to Use SSL

There is no special setup required for LDAP clients using SSL other than the client must have the CAs root certificate in its key database file (see the steps described above). The application must then initiate a secure SSL connection by using the appropriate API calls, that is, `ldap_ssl_client_init()`. If client authentication is configured on the server, the client must be set up with its own certificate as described above for the server.

The application may specify a path to another key database file and may also exploit the password stash file if needed. For more information, see Chapter 8, “Developing Directory-Enabled Applications” on page 223, or the online *Programming Reference* shipped with the product.

5.5 Delegation Model

This section provides a discussion of four alternative solutions that could be used to implement delegation for SSL. Note that delegation is not supported natively by LDAP, and the following discussion, therefore, applies to applications that use LDAP. An example is used to illustrate the four alternatives.

There are several requirements pertaining to the delegation for SSL. The example below that we use throughout this section depicts a typical scenario for delegation (others can be derived from this one). Four solutions are presented to solve the lack of standardized delegation in SSL.

In the example shown in Figure 24 , Mary, on a client machine, uses a Web browser to contact a middle tier server (MTS) using HTTP with SSL (https) over transaction 1 (T1). A program on MTS wants to execute a transaction (T2) on Mary's behalf on an End Tier Server (ETS). The second transaction (T2) should be authenticated as Mary rather than as MTS. On each machine, the SSL code will report to the application the name of the peer. The applications then use this name for access control logging and other security and non-security related purposes. The requirement is that the name that is

reported is the name of the person using the client not the name of the server. This is not possible with current technology without an additional application code.

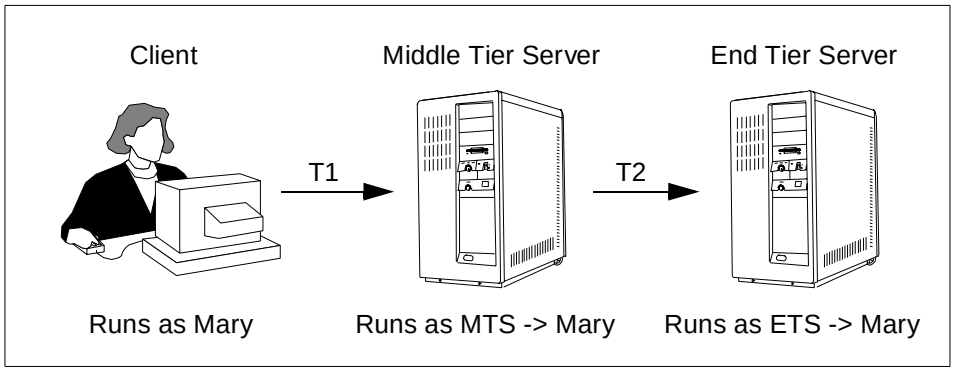


Figure 24. Delegation

The requirement cannot be met because the authentication for T1 and T2 is using SSL client authentication. Although client authentication has changed slightly among SSL Version 2, SSL Version 3, and TLS Version 1.0, all of these protocols have the client sign data derived from the rest of the handshake (session establishment messages). The client (Mary in the T1 transaction in the example above) executes this signature using its private key (normally an RSA key) and then sends the signature and its certificate for this key to the server (MTS for T1 above). This cannot be repeated for T2 because the private key is never permitted to leave the client; so, the MTS cannot get access to it.

This section includes four potential solutions to this problem. All of these solutions assume that a successful client-authenticated SSL or TLS session has been established between the client and the MTS.

Solution 1 - Client creates new certificate
The client extracts the server's name and server's public key from the certificate that the server sent to the client during the handshake. The client then creates a new X509 Version 3 certificate for the server using the name from the client's certificate as the issuer name. The client would normally set the certificate to expire in a very short time. Other restrictions could be added in the form of X.509v3 extensions, but certificate revocation lists (CRLs) would not be used. The client would send the new certificate to the MTS.

The MTS would then start a normal SSL handshake (as a client) with the ETS. When MTS is asked for its certificates, it would send the newly created

certificate it received from the client, and the client's original certificate. ETS would then verify the certificate by constructing a certificate chain, and the rest of the SSL connection for T2 would progress normally.

In this case, the ETS does not know that any delegation is taking place. New code is required on the client and the MTS. The certificate hierarchy for the client must have a policy that allows the client to create valid certificates (as interpreted by the ETS).

Solution 2 - Client create signed document not valid X509 certificate
The client extracts the server's name and server's public key from the certificate that it sent during the handshake as in solution 1 above. In this case, it creates a signed document including its name, the server's name, the server's public key, and any restrictions, such as the end time of the delegated session. Notice that the certificate created in solution 1 would meet this criteria. The signed document is sent to the MTS.

The MTS starts a SSL client-authenticated session but uses its own certificate. During the handshake, it also sends the signed document from the client along with a signal that delegation is to be used, and the ETS should assign the client's identity to this session (T2).

The ETS verifies the SSL handshake in the normal manner then processes the request by the MTS for a delegated session. If the signature on the signed document from the client is valid, the SSL session was set up by the MTS server and public key in the signed document; therefore, the SSL code on the ETS will use the client’s identity.

In this case, the ETS must have modified code and the ETS knows the MTS and client identity. The client certificate must be valid for signature or SSL use (as interpreted by the ETS), but this is a default requirement for client certificates for SSL use.

Solution 3 - Client signs for MTS
The MTS server initiates the SSL session with the ETS and sends the client’s certificate. When the handshake progresses to the point where a signature is required, MTS sends all of the handshake data back to the client. The client (after any desired verification) makes the required signature and sends it to the MTS. The MTS then uses the signature to complete the handshake for T2 with ETS.

In this case, the ETS does not require new code. No special characteristics are required for the client certificate. This solution has the advantage that the client may approve each transaction because it has available the name and

certificate of the ETS and is called for each session set up. On the other hand, this solution has the disadvantage that each MTS/ETS transaction requires traffic between the MTS and the client.

Solution 4 - MTS signs certificate
In this case, the MTS can act as a certificate authority that is trusted (at least by the ETS). After the SSL handshake, the MTS extracts the name from the client certificate, makes up a new RSA key pair, and creates a certificate. It then uses that certificate and private key for the subsequent SSL session with the ETS. The key pair and certificate may be stored for later use.

A variation on this scheme is that the MTS can act as a registration authority for a certificate authority on another machine.

Neither the client nor the ETS need to be modified for this solution. No new SSL (or other related communications flows) are required. Neither the client nor the ETS know that delegation is being done. However, the ETS must be configured to trust either the MTS or a CA that trusts the MTS.

5.6 Access Control

Access Control Lists (ACLs) provide a means to protect information stored in an LDAP directory. Using ACLs, administrators can restrict access to different portions of the directory or specific directory entries. Each entry within the IBM SecureWay Directory has an associated ACL. In conformance with the LDAP model, the directory server stores the ACL information as attribute-value pairs. Furthermore, the LDIF syntax may be used to administer (retrieve/store) these values.

ACL information is broken into two distinct subgroups: The entry owner and the entry ACL. Each directory entry must have both an entry owner and an entry ACL.

Entry owner – The entry owner has complete permissions to perform any operation on the object regardless of the ACL entry. Additionally, the entry owner is the only one, besides the directory administrator, who is permitted to administer the ACL for that object. The entry owner is defined to be an ACL subject.

ACL subjects – The ACL specifically grants a subject permission to perform a given operation. Subjects are considered the combination of a privilege attribute and a distinguished name (DN). Valid types of privilege attributes are:

- Access ID
- Group
- Role

The DN identifies a particular access ID, group, or role. The following are two examples for subjects:

```
access-id: cn=personA, o=IBM, c=US
group: cn=deptXYZ, o=IBM, c=US
```

Not all directory groups may be used in access control. Groups that are to be used in access control must have an objec tclass of AccessGroup and are often called *access groups*. The accessGroup object class is a subclass of the GroupOfNames objectclass.

Another type of privilege attribute used within the ACL model is a *role*. While roles and groups are similar in implementation, conceptually they are different. When a user is assigned to a role, there is an implicit expectation that the necessary authority has already been set up to perform the job associated with that role. With group membership, there is no built-in assumption about what permissions are gained (or lost) by being a member of that group.

Roles are similar to groups in that they are represented in the directory by an object. Additionally, roles contain a group of DNs. Roles that are to be used in access control must have an object class of AccessRole. The AccessRole object class is a subclass of the GroupOfNames object class.

The IBM SecureWay Directory contains two built-in access groups: cn=Anybody and cn=Authenticated. When specified as part of an ACL, this first group refers to all users, and the second refers to all users who are authenticated with the server. Users cannot be removed from these groups, nor can these groups be removed from the database. There is also a built-in access ID: The cn=this. See section 5.6.5, “Pseudo DNs” on page 125, for more information.

5.6.1 ACL Permissions

There are six basic operations that may be performed on a directory object. From these operations, the base set of ACL permissions are taken. These six operations are:

- Add an object
- Delete an object
- Read an attribute value
- Write an attribute value

- Search for an attribute
- Compare an attribute value

These permissions are discrete, that means, one permission does not imply another. The following table, Table 6, describes the permissions needed to perform each of the LDAP operations that are represented by their respective API calls.

Table 6. Permissions required for basic LDAP operations

Operation	Permission Needed
ldap_add()	add (on parent)
ldap_delete()	delete (on object)
ldap_modify()	write (on attribute of object)
ldap_search(), return attribute names	search (on attribute)
ldap_search(), return attribute names and values	search, read
ldap_modrdn()	write
ldap_compare()	compare

The possible attribute class permissions are: read (r), write (w), search (s), and compare (c). Additionally, object permissions apply to the entry as a whole. These permissions are: add child entries (a) and delete this entry (d).

5.6.2 Attribute Classes

Attributes requiring similar permissions for access are grouped together in levels or *classes*. Attributes are mapped to their attribute class in the directory schema file. The three classes available in IBM SecureWay Directory are:

- Normal
- Sensitive
- Critical

Each of these classes are discrete, that means, access to one class does not imply access to another class. Permissions are set with regard to the attribute class as a whole. The permissions set on a particular attribute class apply to all attributes within that class.

5.6.3 Propagation

ACLs and owner can be set to apply to just a particular entry or an entry and the entire subtree. Although both the entryOwner and aclEntry attributes can propagate, their propagation is not linked in any way.

Entries on which an ACL has been placed are considered to have an *explicit ACL*. Similarly, if an owner has been set on a particular entry, that entry has an *explicit owner*. Since the two are not intertwined, an entry with an explicit owner may or may not have an explicit ACL, and an entry with an explicit ACL may or may not have an explicit owner. If these values are not explicitly present on an entry, the values are inherited from an ancestor node in the tree.

Each explicit ACL and owner applies to the entry on which it is set. Additionally, the value may apply to all descendants that do not have an explicitly set value. This values are considered propagated; that is, their values propagate downwards through the directory tree. Propagation of a particular value continues until another propagating value is reached.

An object's ACL (or an object’s owner) can, therefore, be conceptually determined by the following algorithm:

Is there an explicit ACL set at the object?
  If *yes*, then that is the object's ACL.
  If *no*, then traverse the tree backwards until an ancestor node is reached with a propagate ACL.
    If no node is found with a propagate ACL, only the object owner and the administrator will be granted access to the object.

5.6.4 LDAP ACL Attributes

The various aspects of the ACL model are represented in six attributes that are part of every directory entry. These are:

1. entryOwner
2. ownerPropagate
3. aclEntry
4. aclPropagate
5. aclSource
6. ownerSource

The aclSource and ownerSource attributes are not user modifiable. They are maintained by the ACL manager and represent the DN from which this particular entry receives its ACL information. If the source DN is the same as

the object DN, then the ACL has been explicitly set on that object. Otherwise, the ACL is inherited from an ancestor object.

Each of these attributes can be managed using LDIF notation. The following defines the syntax for each of the ACL attributes using Backus-Naur Form (BNF).

```
<subjectSecurityAttribute> ::= "("
                  <subjectSecurityAttributeTypeName> ':'
                  <subjectSecurityAttributeValue>
                  ")"
<subjectSecurityAttributeTypeName> ::= "role" | "group" | "access-id"
<subjectSecurityAttributeValue> ::= <DN>
<entryOwner> ::= "(" <subjectSecurityAttribute> ")"
<aclEntry> ::= "("
              <subjectSecurityAttribute> ":"
              <accessList> [ ":" <accessList> ]
              ")"
<accessList> ::= <objectAccessClass> | <attributeAccessClass>
<objectAccessClass> ::= "object:" <objectAccessClassPermissions>
<objectAccessClassPermissions> ::= "a" | "d"
<attributeAccessClass> ::= <class> ":" <permissions> [ ":" <permissions> ]
<class> ::= "normal" | "sensitive" | "critical"
<permissions> ::=  "r" | "w" | "s" | "c"
<ownerPropagate> ::= "true" | "false"
<ownerSource> ::= <printablestring>
<aclPropagate> ::= "true" | "false"
<aclSource> ::= <printablestring>
```

Example: Default ACL

```
entryOwner:  access-id:cn=admin,c=US
ownerPropagate:  TRUE
aclPropagate:  TRUE
aclEntry:  group:cn=Anybody:normal:rsc
aclSource:  default
ownerSource:  default
```

Example: ACL for entry <cn=personA, ou=deptXYZ, o=IBM, c=US>

```
entryOwner:  access-id:deptXYZMgr, ou=deptXYZ, o=IBM, c=US
ownerPropagate:  TRUE
aclPropagate:  TRUE
aclEntry:  role:cn=Admins, o=IBM,
 c=US:normal:rwcs:sensitive:rwsc:critical:rsc
aclEntry:  group:cn=deptXYZRegs, o=IBM, c=US:normal:rsc:sensitive:rsc
aclEntry:  access-id:cn=personA, ou=deptXYZ,
 o=IBM,c=US:object:ad:normal:rwsc:sensitive:rwsc:critical:rsc
aclEntry:  group:cn=Anybody:normal:rsc
```

```
aclSource:  ou=deptXYZ, o=IBM, c=US
ownerSource:  ou=deptXYZ, o=IBM, c=US
```

The latter example is an inherited ACL and an inherited owner. Both owner properties and ACL properties are inherited from object <ou=deptXYZ, o=IBM, c=US>. In this example, members of group <cn=deptXYZRegs, o=IBM, c=US> have permission to read, search, and compare objects in both the normal and sensitive attribute classes. They do not have permission to add or delete objects under this object, nor do they have permission to access any information or change any information on attributes in the critical class. Unauthenticated and all other bound users have permission to read, search, and compare attributes in the normal attribute class only. PersonA has add and delete permission on the object, read, write, search, and compare permissions on normal and sensitive attributes, and read, search, and compare permission on critical attributes.

5.6.5 Pseudo DNs

The IBM SecureWay Directory contains several pseudo DNs. These are used to refer to large numbers of DNs, which, at bind time, share a common characteristic in relation to either the operation being performed or the object on which the operation is being performed.

The first of these is group:cn=Anybody. When specified as part of an ACL, this group refers to all users, even those that are unauthenticated. Users cannot be removed from this group, and this group cannot be removed from the database.

The second pseudo DN is access-id:cn=this. When specified as part of an ACL, it refers to the bindDn (the DN at which a client binds to the directory) that matches the DN on which the operation is performed. If an operation is performed on the object <cn=personA, ou=IBM, c=US> and the bindDn is <cn=personA, ou=IBM, c=US>, the permissions granted would be the union of those given to <cn=this> and those given to <cn=personA, ou=IBM, c=US>.

The third pseudo DN is group:cn=Authenticated. This DN refers to any DN that has been authenticated by the directory. The method of authentication is irrelevant.

Note

On an add operation, the object whose permission is checked is the parent object; so, the bindDn would have to match the parent object's DN not the DN of the object being added.

5.6.6 Granting Access

Access for a particular operation is granted or denied based on the bindDn (the DN at which a client binds to the directory) for that operation. Processing stops as soon as access has been determined. Within the ACL entry, if there is an aclEntry subject DN that matches the bindDn, the permissions granted are those that are designated by that ACL entry. If there is no aclEntry for that particular bindDn, then group membership is evaluated.

Group and role membership is determined at bind time and is static for the length of the bound connection. The permissions given via group membership are additive. If a user belongs to two groups, both of which are specified within an aclEntry, then the user receives the combined permissions of both of those groups. Additionally, the user will receive any permissions given to the <cn=Anybody> group. If there is no aclEntry for a particular DN, and the subject is not a member of any access groups specified by the aclEntries, then the user receives the permissions given to the <cn=Anybody> group. If there are no permissions specified for this group, permission for the operation is denied.

5.7 Storing Security Related Information in the Directory

Sensitive data can be stored safely in a directory. Sensitive data includes, but is not limited to, passwords, certificates, private keys, and any other data one deems sensitive and should not be readily visible to everyone. The following discussions elaborate on this topic.

5.7.1 Passwords

The most immediate and interesting case is the storing of passwords in the directory service because passwords can be used to authenticate oneself to the directory before data can be accessed. Access to the userPassword attribute (like all other data in the directory) is governed via access control provided by and enforced by the directory server.

By default, transmission of data between the directory client and server is transmitted in the clear. It is, therefore, recommended to use the

ldap_ssl_client_init() and ldap_ssl_init() API calls to set up an SSL connection for privacy during transmission prior to invoking the LDAP APIs.

Passwords show up in (at least) two places in the directory.

- In the person object, where the userPassword attribute is used, for example, for the primary authentication to authenticate oneself to the directory service before accessing data in the directory for primary machine login or for Web server authentication.
- In the account object, where the userPassword attribute can be used for secondary authentication to authenticate oneself to other services. For example, this can be used to authenticate a user/client to other middleware products or to a Web server.

In both cases, for the person object and the account object, the userPassword attribute is used, and there should not be any new password attributes defined.

Let us now look at the two cases (primary and secondary authentication) to understand the use of userPassword. The requirement is the same in both cases: To retrieve the userPassword attribute in a format so it can be used in further operations as expected. How the userPassword attribute is stored by the directory service is orthogonal to transmission. Although userPassword is allowed to be multi-valued, we assume for these examples that the userPassword attribute in the directory only has one value. In the multi-valued case, the values are not ordered; each value of a multi-valued userPassword attribute is checked.

5.7.1.1 Primary Authentication

Primary authentication is the authentication of a client to the LDAP server in order to access (retrieve, store, modify) any data in that directory. This is what is commonly referred to *logging on* to a service.

There are some guidelines for LDAP exploitation projects to provide security for transmission, storage, and access of sensitive information, such as passwords. There are four general guidelines or rules. We recommend that rules A, B, and C should be used. Rule D should not be used, and it is included in this discussion only for completeness.

Rule A: Access to userPassword is governed via access control provided by the directory server. Proper settings of the ACLs must be assured.

Rule B: In the example above, using ldap_bind(), the value of userPassword is transmitted in the clear. The use of ldap_ssl_client_init() and ldap_ssl_init()

prior to the ldap_bind() is recommended. This initiates an SSL connection for privacy during transmission.

Rule C: To mitigate against the threat of disclosure (for privacy), the value of the userPassword attribute is encrypted by the directory server and then stored. The IBM SecureWay Directory supports this option to provide a maximum level of security (see 5.7.1.3, “Password Encryption” on page 128). This method cannot be used when applications need to be able to get a password in clear text.

Rule D: The client application performs a one-way hash operation on the password. It is stored in the directory in its hashed form. Compare operations still work because whatever the user types in is first hashed and then compared against what is stored. It is not recommended to use this method because it requires all applications to agree on a common hash algorithm, which limits the flexibility. Also, as a design point, encryption and password matching should not be done on the application level.

5.7.1.2 Secondary Authentication

Secondary authentication is where an application obtains the user ID and password (userPassword) from the account object to authenticate the user to the system referenced by that account. The password cannot be stored encrypted because the application (most likely) needs a clear password for secondary authentication to the target system or service. Other than with primary authentication, it is the account object that contains the userPassword attribute instead of the person object.

A special case of secondary authentication would be where the LDAP directory itself is the target system for secondary authentication, and the primary authentication is done by a trusted third party service. This would be the case, for example, when the primary authentication is a fingerprint authentication service that provides an identity mapping such that an authentication service can obtain the userPassword from the person object for the person associated with the input fingerprint. The application (or a logon service) can then authenticate the user to the directory using the user ID and the password obtained through the third party authentication service.

5.7.1.3 Password Encryption

The IBM SecureWay Directory V3.1 supports a function where the passwords can be encrypted before they are stored to prevent them from being compromised via direct SQL queries, database file look-ups, or unauthorized accesses. To provide this level of security, the userPassword values can be encrypted in different ways. Two one-way hash algorithms, *SHA* (Secure

Hash Algorithm) and *crypt*, can be selected to encrypt passwords in the permanent store. An SHA-hashed password can only be used for password matching, but it cannot be decrypted. At the user login, the login password is hashed and compared with the stored version for matching verification. Another option for password encryption is by using the two-way encryption option *imask*. With this option, passwords are encrypted before they are stored in the database and decrypted upon retrieval. This prevents clear text passwords from being stored in the database.

Password encryption can be configured through the administration GUI or (though not recommended) by editing the pertinent line in the slapd.conf file. The attribute pwencryption has four options (none | imask | SHA | crypt) to select. The default is imask.

5.7.2 Certificates

Public key certificates are appropriate for storage in a directory. These are needed by anyone wishing to encrypt or sign data to be sent where the receiver uses their private key to decrypt or verify the signature, respectively. Public key certificates are binary data and do not need to be encrypted by the application or the server upon storing in the directory.

Storing private keys in a directory is generally not recommended as they should, by definition, be kept by the owner (for example, stored on a Smartcard). However, if an application chooses to do so, then the rules, as listed above in 5.7.1.1, “Primary Authentication” on page 127, apply.

5.7.3 Displaying Sensitive Data

Clear text passwords should not be displayed on a screen or on paper. The LDAP directory administrative GUI and any other GUIs that deal with passwords that are presented to the user do, or must appropriately, protect any passwords returned when accessing the directory. Appropriate action means something like displaying asterisks (******) in the password field when the password field is displayed on the screen.

5.7.4 Attacks

The encryption methods used to protect secure data must be resistant to attacks based on *stealing* data from the underlying LDAP data store. Two particular attacks are of concern.

- Dictionary attacks, where the attacker encrypts a large set of well-known possible passwords (usually taken from a dictionary) and compares them to encrypted data in the data store.

- Birthday attacks, where the attacker hashes a large number of keys in hopes of finding a matching value in any securely hashed data kept in the data store.

Techniques for foiling these attacks (such as salts, password strength rules, documentation to administrators, and so on) should be employed.

Chapter 6. Installation and Configuration

The IBM SecureWay Directory runs on IBM AIX, OS/400, OS/390, Sun Solaris, and Microsoft Windows NT. This chapter describes the installation and basic server configuration on the IBM platforms. The installation on Sun Solaris is similar to IBM AIX except that the SMIT tool is not available on Solaris. Please read the installation documentation that comes with the product for the most accurate information.

For the latest information and updates about the product on each individual platform, please read the pertinent documentation including the release notes and any *Read Me First* for your platform.

6.1 Windows NT

This section describes the installation and basic configuration of the IBM SecureWay Directory on Microsoft Windows NT. The IBM SecureWay Directory for Windows NT is available on the IBM Suites for Windows NT, and it can be downloaded from the Web (see link below).

For latest information and updates, as well as code downloads, please check the SecureWay Web site:
`http://www.ibm.com/software/network/directory`.

6.1.1 System and Software Requirements

The following are the system and software requirements for the IBM SecureWay Directory server on Windows NT.

- Windows NT 4.0 with:
  - Service Pack 3 (or greater).
  - An NTFS partition.
  - A minimum of 64 MB of memory is recommended.
- One of the following Web servers must be installed and configured:
  - IBM HTTP Server powered by Apache 1.3.3.1
  - Apache 1.3.2 or later
  - Lotus Domino Go 4.6.2 or later
  - Microsoft Internet Information Server 2.0
  - Netscape FastTrack 3.01
  - Netscape Enterprise 3.5.1

  (A Web server is necessary to run the Web-based configuration and administration tools.)

- A Web browser that supports the following:
  - Frames.
  - HTML version 3.0 or later.
  - Java 1.1.7 features including JDK 1.1 AWT events.
  - JavaScript 1.2.
  - The browser must be enabled to accept cookies.

  Most recent versions of popular Web browsers, including Netscape Navigator/Communicator 4.X or Microsoft Internet Explorer 4.X. and 5.X, are sufficient.
- DB2 Workgroup, Personal (Universal Personal Edition), or Enterprise edition, Version 5.2, with fixpack WR09084.
- Approximately 70 MB of disk space for the IBM SecureWay Directory including DB2. If DB2 is already installed, approximately 25 MB is needed for the IBM SecureWay Directory server.

6.1.2 Installing the Server

Depending on the source from where the IBM SecureWay Directory was obtained, you may have to unpack (unzip) a single file first. The IBM SecureWay Directory uses the Windows InstallShield that can be launched by running setup.exe from within the directory where the IBM SecureWay Directory files reside.

Before installing the IBM SecureWay Directory, the Web server and DB2 must be installed and running. Run the respective installation process for these components. If you install the Personal edition of DB2, make sure that you do *not* select the Connect Personal edition; the IBM SecureWay Directory requires the Universal Personal edition.

With the Web server and DB2 installed and running, run the setup.exe from the IBM SecureWay Directory, and the InstallShield will guide you through the installation process. The InstallShield allows you, at the same time, to do some basic configuration of the server. However, this basic configuration can also be done or changed later using the `ldapxcfg` command. An advantage of running the configuration using `ldapxcfg` after the actual installation is that you will be able, at that time, to access the online documentation.

The installation using the Windows InstallShield covers the following four general steps:

1. Language selection – You can choose one from the supported ten languages (English, French, German, Japanese, Simplified Chinese,

Traditional Chinese, Italian, Spanish, Brazilian Portuguese, and Korean). The default is English.

2. Select destination location – Choose the destination directory for the installation. The default is C:\Program Files\IBM\LDAP\.
3. Select the components to configure – A selection allows you to select:
   - Set the directory administrator name and password
   - Create the directory DB2 database
   - Configure a Web server for directory administration

   See the next section 6.1.3, “Configuration” on page 133, for more details about configuration of the IBM SecureWay Directory server.
4. Restart the computer – The system needs to be restarted after installing the IBM SecureWay Directory.

As mentioned above, the configuration (step three from above) can also be done after the installation. It is explained in the section that follows. The database configuration can also be done through the administrator GUI, which is explained in 7.3, “Database Configuration” on page 169.

The online documentation can be accessed after installation using a Web browser as follows where *x:**<inst_dir>* is the root installation directory (default: C:\Program Files\IBM\LDAP\), and *<language>* is the installation language directory (default: enUS1252 for English):

The *Installation and Configuration Guide*:
x:\<inst_dir>\nls\html\<language>\config\wparent.htm

The *Administration Help*:
x:\<inst_dir>\web\<language>\help\parent.htm

The *Directory Management Tool*:
x:\<inst_dir>\nls\html\<language>\dmt\dparent.htm

The *C Programming Reference*:
x:\<inst_dir>\doc\progref.htm

The *JNDI Programming Guide*:
Unzip x:\<inst_dir>\java\ibmjndi.zip and load ibmjndi\Guide.html

6.1.3 Configuration

This section describes the third step from the installation above, which you are either prompted for during the InstallShield installation or by using the ldapxcfg configuration utility after the installation. The following discussion is based on ldapxcfg, but it equally applies to the InstallShield configuration as well.

You can either launch the ldapxcfg utility manually or double-click the **SecureWay Directory Configuration** icon in the IBM SecureWay Directory folder on the desktop. There are three options to configure as shown in Figure 25 below.

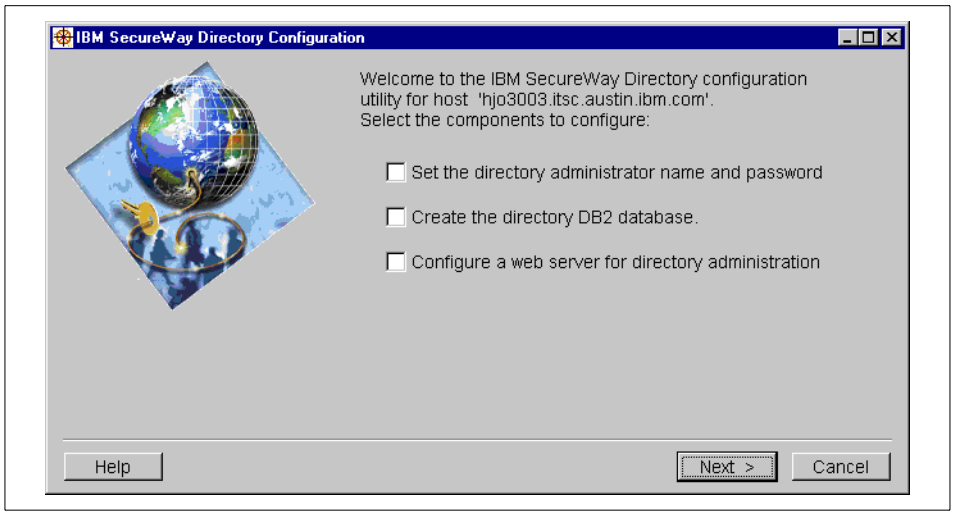


Figure 25. IBM SecureWay directory configuration (using ldapxcfg)

The installation and configuration utility determines if it is a new installation or an upgrade from an earlier version to the current version 3.1. If it is an upgrade, a prompt window will ask you whether you want to migrate the directory. If you choose to migrate, the migration will be done automatically. Such a migration concerns the flat files containing schema definitions and the configuration file; the database does not need to be migrated (see also 4.6, “Migration from the Previous Release” on page 91). The old schema files will be kept, and the old configuration file will be renamed to slapd21.conf.

On the panel shown in Figure 25, you have three options to configure; they can be done all at once by selecting all options or one after the other, but all must be done.

- To set the directory administrator name (DN) and password:
  1. Select **Set the LDAP administrator name and password** and click the **Next >** button.
  2. Type in the administrator DN (or accept the default DN) and type in a password and proceed.
- To create the DB2 database:

1. Select **Create the directory DB2 database** and click the **Next >** button.

> **Note**
>
> Before you move on, you may want to browse through the database configuration description in 7.3, “Database Configuration” on page 169. It describes the database creation through the administrator GUI. The basic considerations, however, are the same and apply to this section as well.

2. The installation and configuration utility will ask you if you want to use the default LDAP DB2 database for the directory server or if you want to configure the directory server to use an existing database. The utility determines if an LDAP database currently exists. If you choose to create a new database while an LDAP database already exists, all data will be lost. It is, therefore, highly recommended to back up an existing database, for example, by using the db2ldif utility. Select whether you want to create a new, or use an existing database.
3. If you choose to use a default database, a new database instance will be created. The default database may be adapted to support UTF-8 (see 4.3, “UTF-8 Support” on page 83). If you plan to use this IBM SecureWay Directory feature, you have to select **Create the default UCS-2 DB2 database (UTF-8)**.
4. If your choice is to use an existing database, you have to provide the following information about this existing database:
   - Database name
   - Database instance
   - Database system administrator ID
   - Database system administrator password

> **Note**
>
> The database name ldapdb2 is reserved by the IBM SecureWay Directory. Do not use it for your own database name.

- To configure a Web server:
  1. Ensure that the Web server is installed and configured.
  2. Select **Configure a web server for directory administration** and click the **Next >** button.
  3. Choose the respective server from the list and click the **Next >** button.

4. Type the name (or accept the proposed path and name if it is correct) of the configuration file of your server and click the **Next >** button to proceed.
5. Do not forget to restart the Web server after its configuration has been changed for these changes to take effect.

The ldapxcfg utility gathers all information that is needed to complete each of the above three tasks before presenting a summary screen and requesting a final click on **Configure** before it executes the configuration change.

Tip

The online documentation titled *Installation and Configuration Guide* contains additional information about the installation of the database and the IBM SecureWay Directory. Please also read any Release Notes or *Read Me First* documents that you may have received with the product.

If you also need SSL for secured LDAP communication, you will have to install and configure the IBM GSKit. Please read 5.3, “SSL Utilities” on page 108 and 5.4, “Configuring SSL Security” on page 111 for further information.

After installation and configuration, the IBM SecureWay Directory server is ready to be started. If not already done, a database must be created (or configured), and then at least one suffix must be defined before any data can be imported. This can conveniently be done through the administration GUI. Please go to Chapter 7, “LDAP Data and System Administration” on page 159 and follow the directions provided.

Command Line Configuration

As an alternative method of using the graphical ldapxcfg configuration utility, there is also an ldapcfg command line tool that can be used for the IBM SecureWay Directory configuration. This is especially helpful when automated configuration methods need to be developed. Please read the instructions in the online *Installation and Configuration Guide* or run the command without parameters to get the command syntax help text.

6.1.4 Unconfiguring and Uninstalling the Server

Before you remove the IBM SecureWay Directory server from a Windows NT system, it is recommended that its configuration be removed from the Web server first. This can be accomplished with the `ldapucfg` command as in this example:

```
ldapucfg -s apache -f c:\apache\conf\srm.conf
```

The example above specifies Apache as the Web server. The path name of the configuration file will likely be different in other installations. Please read the help text that is displayed when issuing `ldapucfg -?` for other options and parameters pertinent to other Web servers. Note that the Web server most likely needs to be restarted (depending on the particular server) in order to reread the modified configuration.

The -d option of the `ldapucfg` command optionally removes a default database instance for LDAP if one was created during configuration (a custom database, however, will not be removed). This option also removes configuration information from the configuration file slapd.conf.

To uninstall and remove the IBM SecureWay Directory server, you may either use the Windows NT Add/Remove Programs function from the Control Panel or select the **unInstallShield** function from the Start -> IBM SecureWay Directory menu list.

6.2 AIX

At the time of writing, the IBM SecureWay Directory V3.1 for AIX was not available yet, and packaging information was not finalized. Though this is not a formal announcement, it can be assumed that the IBM SecureWay Directory will be available on either the AIX product CD-ROMs or the AIX Bonus Pack CD-ROMs. Additionally, the IBM SecureWay Directory might be made available for download from an IBM Web site (see link below). According to current planning at the time of writing, a version of DB2 that is suitable for running the LDAP server will be included with the IBM SecureWay Directory.

For latest updates and product information, you should always check the Web site
`http://www.ibm.com/software/network/directory`

or the latest AIX announcements at:
`http://www.ibm.com/rs6000/software`

The following is a description of the installation and configuration that is based on planning information rather than actual product code. All information provided below is, therefore, subject to change.

6.2.1 System and Software Requirements

The following are the system and software requirements for the IBM SecureWay Directory server on IBM AIX:

- AIX Version 4.2.X or 4.3.X
- A Web browser that supports the following:
  - Frames.
  - HTML version 3.0 or later.
  - Java 1.1.7 features including AWT events.
  - JavaScript 1.2.
  - The browser must be enabled to accept cookies.

  Most modern Web browsers, including Netscape Navigator/Communicator 4.X or Microsoft Internet Explorer 4.X or 5.X, meet these requirements.
- One of the following Web servers must be installed and configured:
  - Apache 1.3.3 or later (or the IBM HTTP Server Powered by Apache)
  - Lotus Domino Go 4.6.2 or later
  - Netscape FastTrack Server version 2.0.1 or later
- DB2 Version 5.2 or later (a suitable version will likely be available with the IBM SecureWay Directory).
- Java JDK (included with the AIX CD-ROMs).
- A minimum of 64 MB memory (128 MB or more is recommended).
- Approximately 77 MB of disk space for the IBM SecureWay Directory and DB2. If DB2 is already installed, 25 MB is needed for the IBM SecureWay Directory.

6.2.2 Installing the Server

As a prerequisite to installing the IBM SecureWay Directory, make sure you have a suitable Web server installed and configured (see the list of supported Web servers above).

The IBM SecureWay Directory comes as a set of installable AIX filesets that can most conveniently be installed using SMIT (Systems Management Interface Tool). As a root user, and with the IBM SecureWay Directory filesets available either on disk or on a CD-ROM, do the following:

1. Run: `smitty install_latest` (or `smit install_latest` for the Motif version of SMIT).
2. Select the appropriate input device or file system directory, such as /dev/cd0.

3. With the cursor in the SOFTWARE to install field, press **F4** to list the installable options available from the source specified. Select the options you want to install and make sure you have at least the LDAP server and the appropriate HTML documentation selected.
4. It is recommended to have AUTOMATICALLY install requisite software set to **YES** (which is the default) in order to install any required additional software, such as DB2 or the LDAP client, automatically.
5. Optionally set PREVIEW only to **YES** and run a preview of the installation first to check for any potential errors or verify the list of options to be installed.
6. Check and set the other options as appropriate and press **Enter** to start the installation.

This installs the IBM SecureWay Directory server and its prerequisite software options, such as DB2, LDAP client, or message catalog filesets, on your system.

6.2.3 Configuration

The configuration of the IBM SecureWay Directory server involves three steps:

1. Defining an administrator ID and password for the LDAP server.
2. Creation of a DB2 database for LDAP data storage.
3. Configuration of the Web server to make the Web based configuration tool work.

These configuration steps can be done using the `ldapcfg` command line tool or using the graphical ldapxcfg utility. Their use is identical to the same tools provided for the Windows NT version of the IBM SecureWay Directory server. Please read the appropriate section 6.1.3, “Configuration” on page 133, as it equally applies for AIX (subject to change).

6.2.4 Unconfiguring and Uninstalling the Server

Before removing the IBM SecureWay Directory, it is recommended to remove the current configuration from the Web server. Type on the command line:

```
# ldapucfg -s <server type> -f <full path of configuration file>
```

Once the unconfiguration is done, the product can be uninstalled. If you want to keep your installation, and just want to restart with a new configuration, take care of the slapd.conf file, remove if it presents the lines about adminPW

and adminDN on the top of the file and any lines about DB2 below database rdbm:

```
suffix        "o=ibm_uk,c=uk"
databaseName  ldapdb2
dbInstance    ldapdb2
dbuserpw      <secret>
dbuserid      ldapdb2
```

Uninstallation of the IBM SecureWay Directory can then be done using the SMIT remove software products option (fastpath: `smitty install_remove`).

6.3 OS/390

The LDAP server on OS/390 is shipped with the OS/390 Security Server and uses DB2 as its backing store. In general, the LDAP server that runs on OS/390 accepts the LDAPv2 protocol. The server, like the AIX and Windows NT LDAP implementation, supports referrals and is capable of replicating its information to other LDAP-capable servers.

There is one difference between the two implementations worth noting. Directory entry data is stored in the DB2 backing store in so-called wire format. Thus, whatever information is stored into the Directory will be returned as it was entered. However, in order to support attribute-based searching, character string data is converted to the local code page that the LDAP server is running in on the OS/390 system. This is typically an EBCDIC code page. During this conversion, any characters not representable by the local code page are translated with some loss of information during the translation. This will affect attribute-based searches for entries where the character string attribute values contain characters not representable in the EBCDIC code page.

This section gives an overview of the LDAP server installation on OS/390 Version 2, Release 7. It is based on information from a previous ITSO redbook: *Ready for e-business: OS/390 Security Server Enhancements*, SG24-5158, including the updates brought by OS/390 V2R7. For detailed instructions for installation and configuration of LDAP on OS/390, please refer to the *OS/390 V2R7 Security Server LDAP Server Administration and Usage Guide*, SC24-5861.

The following list contains an overall description of the LDAP features and functions on different OS/390 releases:

OS/390 R4 (GA 9/97)

- LDAP V2 C language client

OS/390 R5 (GA 3/98)

- OS/390 R4 function
- LDAP V2 Server (based on DB2 backing store, common code base with IBM SecureWay Directory)

OS/390 R6 (GA 9/98)

- OS/390 R5 function
- LDAP V3 C language client
- Remote ACL administration via IBM command-line LDAP ACL utility (ldapcp)
- Allow multiple LDAP servers to run on the same OS/390 image (independently and must use separate databases)

OS/390 R7 (GA 3/99)

- OS/390 R6 function.
- JNDI, with a single SPI, for LDAP.
- Access to RACF USER and GROUP profiles.
- Sysplex support allows multiple LDAP servers to access and update the same set of DB2 tables when used with DB2 data sharing.

6.3.1 System and Software Requirements

The OS/390 LDAP server is part of the OS/390 Security Server feature, which is linked with the V2R7 Release of OS/390. As such, LDAP has the same system and software prerequisites as the OS/390 Security Server. In addition, DB2 Version 5 must be installed.

For more information on OS/390 and its installation prerequisites, please go to:
`http://www.ibm.com/s390/os390/installation`

6.3.2 Installing the Server

The LDAP server installation is a normal SMP/E installation of yet another product into your system. You would, of course, have verified that all relevant maintenance has been installed and that you have reviewed the *Memo to Users* to check for any late changes. See also the *OS/390 Security Server Program Directory*, which comes with the LDAP server tape or cartridge.

In OS/390 Release 7, all DLLs for the LDAP server are now shipped in PDS format only. In order for these DLLs to be located by the server at runtime, the PDS, which contains these DLLs (*GLDHLQ*.SGLDLNK, where *GLDHLQ* is the

high-level qualifier used when installing LDAP code), must either be in the *LINKLIST*, referenced in a *STEPLIB DD* card (if the LDAP server is started from JCL), or listed in the *STEPLIB* environment variable (if the LDAP server is started from the OMVS command prompt). Any of these methods can be used, and the choice of the best method is dependent of the reader's operations procedures in his/her environment.

In OS/390 Release 7, access to RACF information through the LDAP server has been added. The PDS, which contains the LDAP server and the DLLs, must now be APF-authorized to allow the LDAP server to make the RACF calls necessary to provide this access. Also, if program control is active on your system, the PDS, which contains the LDAP server and the DLLs, and the PDS that contains the C runtime libraries and SYS1.LINKLIB, must be program controlled. It may also be necessary to APF-authorize and program control the DB2 SDSNLOAD dataset, which contains the DLL loaded to make the DB2 CLI calls.

6.3.3 Configuration

After successful installation of the LDAP code, you should prepare for running the server on your system. You have to ensure that the necessary prerequisites are in place and that you take necessary actions prior to starting your server. Assuming you use the OS/390 Security Server, the following list describes the necessary preparatory work you should do:

- Define a user ID for running your LDAP server.
- Ensure that you have DB2 installed and set up for running the Call Level Interface (CLI) and Open Database Connectivity (ODBC).
- Create the LDAP server DB2 database.
- Create the LDAP server configuration file.

These steps are described in the following section.

6.3.3.1 Define a User ID for Running the LDAP Server

You need to define a RACF user ID under which to run the LDAP server. This user ID has to be a so-called superuser (UID equals zero). In addition, you need to give this user ID relevant access to the FACILITY class profiles (BPX.DAEMON and BPX.SERVER) if you are using thread-level security. Finally, you would have to permit the user ID access to the MVS data sets defined in the LDAP procedure JCL.

The following sample RACF commands can be used to define LDAPSRV:

```
ADDGROUP LDAPGRP SUPGROUP(SYS1) OMVS(GID(20))
```

```
ADDUSER LDAPSRV DFLTGRP(LDAPGRP) OMVS(UID(0) PROGRAM('/bin/sh'))
PERMIT BPX.DAEMON CLASS(FACILITY) ID(LDAPSRV) ACCESS(READ)
PERMIT BPX.SERVER CLASS(FACILITY) ID(LDAPSRV) ACCESS(UPDATE)
```

You also need to define a user ID for the procedure that will be used to start the LDAP server. The following commands can be used:

```
RDEFINE STARTED LDAPSRV.** STDATA(USER(LDAPSRV))
SETROPTS RACLIST(STARTED) REFRESH
```

6.3.3.2 Setting Up DB2 to Run the LDAP Server

The LDAP server requires you to have DB2 Version 5 installed on the system. Make sure the necessary maintenance for the LDAP server is installed because there is a specific PTF level for DB2 V5 required (PQ09901 is the APAR number). Edit and submit the job in *GLDHLQ*.SDSNSAMP(DSNTIJCL), where *GLDHLQ* is the high-level qualifier used for your DB2 installation data sets. See the section on setting up the DB2 Call Level Interface (CLI) runtime environment in *DB2 for OS/390 Call Level Interface Guide and Reference*, SC26-8959. You must run the job under a user ID with the necessary database authority. If you have problems finding the job in your DB2 sample library, you are probably missing some required PTFs.

The next task is to create a DB2 CLI initialization file. There is a sample file in *GLDHLQ*.SDSNSAMP(DSNAOINI) that you should copy into a sequential dataset with a name of your choice. (Note: this must be a dataset not a sequential file in HFS. Also, the dataset must be FB 80.) Again, you can find out more about the contents of this file in *DB2 for OS/390 Call Level Interface Guide and Reference*, SC26-8959.

Figure 26 shows a sample DSNAOINI file.

```
; This is a comment line...
; Example COMMON stanza
[COMMON]
MVSDEFAULTSSID=yoursubsystemname
; Example SUBSYSTEM stanza for your DB2 subsystem name
[yoursubsystemname]
MVSATTACHTYPE=yourmvsattachtype
PLANNAME=yourCLIplanname
; Example DATA SOURCE stanza for your data source
[yourdatasourcename]
AUTOCOMMIT=0
CONNECTTYPE=1
```

Figure 26. Sample DSNAOINI file

The term DATA SOURCE may be a bit confusing in tailoring the DSNAOINI file to your specifications. The other name for this parameter is DB2 server location, which may be easier to relate to. Be careful when you edit the file so that you do not inadvertently destroy the square brackets in your initialization file.

6.3.3.3 Creating the LDAP Server DB2 Database and Table Spaces

There is a sample script in *GLDHLQ*.SGLDSAMP(LDAPSPFI) that you should tailor with your own resource names and then run the SPUFI (SQL Processor Using File Input) script from the DB2 Interactive (DB2I). DB2I is a DB2 facility that provides for the running of SQL statements, DB2 (operator) commands, and utility invocation.

If you have an existing R5 or R6 LDAP BD2 database, you can migrate your database using the ldapspfi.spufi.migrate file (LDAPSPMG in the sample PDS). The recommendation is that you merge the information in this file with the original sample file.

You should run this sample script under a user ID that has the SYSADM authority. Depending on how you have set up your DB2 system, the double minus signs may not be interpreted as comments, in which case, you have to change them to your own standard (Figure 27).

```
--//*************************************************************//
--//*
--//* Licensed Materials - Property of IBM
--//* 5647-A01
--//* (C) Copyright IBM Corp. 1997
--//*
--//*************************************************************//
-- Use the following statements to create your LDAP Server DB2 database
-- and tablespaces in SPUFI. The database and tablespace names you
-- create will be used to update the database section of the LDAP
-- Server configuration file.
-- Change dddddddd to the name of the LDAP database name you want to create.
-- Change the aaaaaaaa to the LDAP entry tablespace name you want to create.
-- Change the bbbbbbbb to the LDAP 4K tablespace name you want to create.
-- Change the cccccccc to the LDAP 32K tablespace name you want to create.
-- Change the eeeeeeee to another LDAP 4K tablespace name you want to create.
create database dddddddd;
create large tablespace aaaaaaaa in dddddddd numparts 1 bufferpool BP32K;
create tablespace bbbbbbbb in dddddddd segsize 4 bufferpool BP0;
create tablespace cccccccc in dddddddd segsize 4 bufferpool BP32K;
create tablespace eeeeeeee in dddddddd locksize tablespace bufferpool BP0;
-- Use the following statements if you need to delete your LDAP Server DB2
-- database and tablespaces in SPUFI. You need to remove the "--"
-- from each line before you can run these statements.
-- Change dddddddd with the name of the LDAP database name you want to delete.
-- Change the aaaaaaaa to the LDAP entry tablespace name you want to delete.
-- Change the bbbbbbbb to the LDAP 4K tablespace name you want to delete.
-- Change the cccccccc to the LDAP 32K tablespace name you want to delete.
-- Change the eeeeeeee to another LDAP 4K tablespace name you want to delete.
--drop tablespace dddddddd.aaaaaaaa;
--drop tablespace dddddddd.bbbbbbbb;
--drop tablespace dddddddd.cccccccc;
--drop tablespace dddddddd.eeeeeeee;
--drop database dddddddd;
```

Figure 27. Sample ldapspfi.spufi file

Save the LDAPSPFI member so that you can use it to delete (drop) the tables when it becomes necessary (change the CREATE commands into comments and remove the comment characters (--) from the corresponding DROP commands).

6.3.3.4 Creating the LDAP Server Configuration File

The LDAP server is highly configurable through a configuration file that allows you to tailor most aspects of the server. The UNIX implementations of the LDAP server call it the Stand-alone LDAP Daemon or SLAPD, which is why the server itself and all the HFS configuration files for it use that name.

The sample configuration file

The default name for the directory used to install the LDAP server is /etc/ldap. The sample directory for the configuration file and other dependent files is

found in the /usr/lpp/ldap/examples/sample_server directory. The /etc/ldap directory is used as the name for the production directory. If you use a different name, you must create symbolic links from the appropriate files in your directory to the /etc/ldap directory. You should also make sure to save copies of the original files prior to your starting to modify them for production use.

To make your own configuration file, you could create a file, such as myslapd.conf, and copy the contents of the default /etc/ldap/slapd.conf into it. Note that the sample slapd.conf file assumes a simple configuration, and if you want to make use of other capabilities (for example, use of multiserver, use of sysplex support), you should read Chapter 6, "Configuring" in the *OS/390 V2R7 Security Server LDAP Server Administration and Usage Guide*, SC24-5861, for more detailed descriptions on how to tailor the slapd.conf file. Figure 28 shows a sample file.

There are a couple of points worth noting in the way you specify values for some of the parameters in the configuration file. First, when in doubt, you should specify the full path name for your include files and the like. The other point is the way in which you specify the name for the dsnaoini parameter when you use an MVS data set. Note that you do not use any quotes or extra indicators, just the fully qualified MVS data set name.

```
#/************************************************************
# * This file is shipped in code page IBM-1047 and must remain
# * in code page IBM-1047.
# ************************************************************/
#/************************************************************
#*
# * Licensed Materials - Property of IBM
# * 5647-A01
# * (C) Copyright IBM Corp. 1997, 1998
# *
# ************************************************************/
#/************************************************************
# * Filename slapd.conf
# *
# * This file is the LDAP Server configuration file for OS/390.
# ************************************************************/
#referral ldap://ldap.itd.umich.edu
include /etc/ldap/slapd.at.system
include /etc/ldap/slapd.at.conf
include /etc/ldap/slapd.oc.system
include /etc/ldap/slapd.oc.conf
port 389
securePort 636
security none
sslKeyRingFile /etc/keyfile.kyr
sslKeyRingFilePW xxxxxx
sslCipherSpecs 12288
maxthreads 0
maxconnections 0
waitingthreads 0
timelimit 3600
sizelimit 500
# The following adminDN and adminPW options should be updated with
# appropriate values. Remove the '#' to uncomment these options.
adminDN "cn=LDAP Admin,ou=Austin,o=IBM,c=US"
adminPW xxxxxx
#########################################################################
# rdbm database definitions
#########################################################################
database rdbm GLDBRDBM
# The following options must be filled in with appropriate values
# for your DB2 setup, prior to attempting to run with the DB2 backend.
servername STLEC1
databasename LDAPDB
dbuserid HILDING
tbspaceentry LDAPTBSP
tbspace32k LDAP32K
tbspace4k LDAP4K
dsnaoini HILDING.ICF.DSNAOINI
suffix "cn=localhost"
suffix "o=IBM_US,c=US"
index cn eq,sub
index ou eq,sub
index sn eq,sub
index telephoneNumber eq,sub
index title eq,sub
readOnly off
```

Figure 28. A sample SLAPD file

6.3.3.5 Grant DB2 Resource Authorizations

If a separate user is created to run the LDAP server, such as LDAPSRV as used in the previous steps, you must grant this user ID the necessary authorizations for DB2. Use the following commands, where <planname> is the CLI plan name as specified in the DB2 CLI initialization file (see 6.3.3.2, "Setting Up DB2 to Run the LDAP Server" on page 143), and <dbname> is the

name of the database as defined in SLAPD configuration file (see 6.3.3.4, “Creating the LDAP Server Configuration File” on page 145).

```
grant execute on plan <planname> to LDAPSRV
grant select on sysibm.systables to LDAPSRV
grand dbadm on database <dbname> to LDAPSRV
```

6.3.3.6 Creating Entries in the SLAPD Database

Your LDAP directory has to be populated with the entries that you want to be able to look up in the applications that are going to use the directory. For this reason, you need to create an LDIF file and run the ldif2db utility program to populate your LDAP directory. You can find a sample LDIF file, sample.ldif, in /usr/lpp/ldap/examples/sample_server. Add an additional suffix in your slapd.conf file, as shown in the example in Figure 28, where a suffix <o=IBM_US,c=US> has been added. Do not remove the suffix <cn=localhost>.

Keep in mind that, if your directory entries have information in them that is not defined in the default directory schema, you have to update the default files as described in “Directory Schema” of the *OS/390 V2R7 Security Server LDAP Server Administration and Usage Guide*, SC24-5861.

6.3.3.7 Starting the LDAP Server

With all the configuration and DB2 work done, you are now ready to start the LDAP server. You can either start it from UNIX Systems Services (USS) or by using JCL. There are a number of things that have to be addressed, however, and they are not pointed out all that clearly in the documentation. You may want to use the following checklist when you start working with the LDAP server:

- Environment variables
  - PATH
  - LIBPATH
  - NLSPATH
  - MANPATH
  - _CEE_ENVFILE
  - LDAP_BASEDN
- TCP/IP configuration
  - RESOLVER_CONFIG
- Language specifications
  - LANG

Environment variables

Especially if you are already running other USS applications, such as a Web server, you have set up your specifications for your working environment accordingly. With the LDAP server as an addition to your existing environment, you have to add the new paths into the existing ones. If your path for help text is specified as MANPATH=/usr/man/%L, you may start experiencing results like the following:

```
# man ls
No Manual entry for "ls"
```

There are various ways of fixing the preceding problem, but one of the simplest is to change the environment variable to MANPATH=/usr/man/C. In working with the LDAP server, the following environment variables were found to be quite useful not only for running the LDAP server but also for running tools like ldapsearch:

```
PATH=/bin:/usr/bin:/usr/sbin:/usr/lpp/ldap/sbin:.
LIBPATH=/lib:/usr/lib:.
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lpp/ldap/lib/nls/msg/%L/%N
MANPATH=/usr/man/C
LANG=En_US.IBM-1047
_CEE_ENVFILE=/etc/ldap/slapd.envvars
LDAP_BASEDN=o=IBM_US,c=US
RESOLVER_CONFIG=//'TCPIP.INTRA.TCPPARMS(TCPDATA)'
```

The various PATH variables are specifying multiple paths with a colon between one path and the next. Starting with OS/390 R7, the setting of the LANG variable can also be C instead of En_US.IBM-1047.The LDAP_BASEDN does the same thing as the suffix <o=IBM_US,c=US> in the LDAP configuration file. The RESOLVER_CONFIG variable is just used to tell TCP/IP where the profile information is to be found. The main use for these variables is to make it simpler to run the various LDAP utility programs, such as ldapadd, ldapmodify and ldapsearch.

Starting the server from HFS

Depending on your settings for PATH, as discussed above, you can start `slapd` in one of the following ways:

```
# /usr/sbin/slapd -f myslapd.conf
```

or

```
# slapd -f /etc/ldap/myslapd.conf
```

The `-f myslapd.conf` parameter denotes the configuration file you previously created.

Running the server as a started task

There is a sample procedure for running the LDAP server as a started task in *GLDHLQ*.SGLDSAMP(LDAPSRV), where *GLDHLQ* is the high-level qualifier used when installing the LDAP code. Customize the *LDAPSRV* JCL and start the job from console or SDSF. Also be sure to have STEPLIB setup to reference *GLDHLQ*.SGLDLINK before starting the job.

When `SLAPD` has been started and is ready, the message:

```
GLD0122I SLAPD is ready for requests
```

is displayed.

The SYSTCPD DD card is an addition that might be useful to help the LDAP server resolve some TCP/IP name resolution tasks.

6.3.4 Unconfiguring and Uninstalling the Server

Because the LDAP server installation is a normal SMP/E function, unconfiguring and uninstalling LDAP will use the same procedure as any SMP/E uninstallation would do.

6.4 OS/400

In general, the LDAP DB2 server that runs on OS/400 supports the LDAP V2 protocol. The server supports referrals and is capable of replicating its information to other LDAP-capable servers. Replication is done using the LDAP protocol.

Similar to the LDAP server for OS/390, there is one difference between the OS/400 and AIX implementations worth noting. Directory entry data is stored in the DB2 backing store in so-called wire format. Thus, whatever information is stored into the directory will be returned as it was entered. However, in order to support attribute-based searching, character string data is converted to code page 37, which contains the IA5 character set supported by LDAP V2. During this conversion, any characters not representable by local code page 37 are translated with some loss of information during the translation. This will affect attribute-based searches for entries where the entry data contains these characters.

The OS/400 directory server is currently based on LDAP V2 (time of writing). It utilizes DB2/400 to store LDAP information, uses a new ACL (Access Control List) design to protect LDAP entries, and uses the Operations Navigator for administration.

The OS/400 LDAP utilities include shell utilities to search, add, delete, and modify LDAP entries. Symbolic links that point to these are created so that they can be easily used in the OS/400 QShell Interpreter (qsh). It also includes interchange utilities to exchange data between the AS/400 and other servers via LDIF (LDAP Data Interchange Format).

6.4.1 System and Software Requirements

The following are the system and software requirements for LDAP server on AS/400:

- OS/400 V4R3 or later
- Option 32 of Operating System/400, OS/400 Directory Services, which includes:
  - OS/400 LDAP Directory Server
  - OS/400 LDAP Client APIs & Utilities
  - Publishing APIs and GUI
  - Windows 9x/NT Client
- Option 30 of Operating System/400, the OS/400 QShell Interpreter
- 5769-JV1, AS/400 Developer Kit for Java to utilize Sun's Java Naming and Directory Interface (JNDI)

6.4.2 Installing the Server

Before you can configure the directory server, you must install the Directory Services option of OS/400. Directory services may already be installed on your AS/400. To install the Directory services option, take these steps:

1. Ensure you have both *ALLOBJ and *IOSYSCFG special authorities.
2. Insert the CD-ROM that contains OS/400.
3. Type `GO LICPGM` on the AS/400 command line, then press **Enter**.
4. Choose option **1** from the Work with Licensed Programs menu, then press **Enter**.
5. Enter **1** in the Option field to the left of Option 32, OS/400 - Directory Services, then press **Enter**.
6. In the Installation device field, enter the name of the CD-ROM drive where you inserted the OS/400 CD-ROM.
7. Press **Enter** to proceed and complete the installation.

6.4.3 Configuration

There is no command line interface for configuring the directory server. The AS/400 Directory Services provides a wizard in the Operations Navigator to assist you in configuring the LDAP directory server. Use this wizard when you initially configure the directory server. You may also use the wizard to reconfigure the directory server.

The Operations Navigator lets you configure the LDAP server on the AS/400 system. The configuration task is one of the network server tasks. You must have both the *ALLOBJ and *IOSYSCFG special authorities to be able to run the configuration wizard. To configure the LDAP server, click **Network** (A) **--> Servers** (B) **--> TCP/IP** (C), as shown in Figure 29. Then, right-click on **Directory** (D). If this is the first time configuration is being run, the pop-up menu will appear as shown at the bottom in Figure 29. If this LDAP server had been configured before, the LDAP Administration Pop-Up menu will appear as shown in Figure 30.

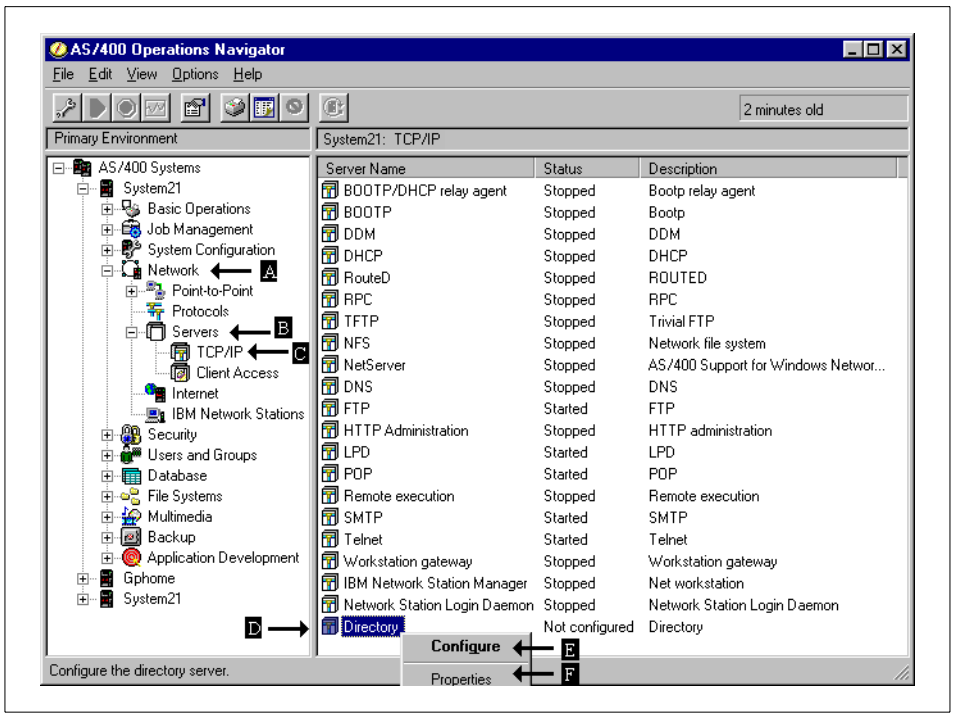


Figure 29. Operations navigator LDAP administration

If this is the first time to configure the LDAP server on your AS/400, click **Configure** (E in Figure 29) to launch the configuration wizard as shown in

Figure 31, which will guide you through the configuration task. If the LDAP server was already configured, click **Reconfigure** (G in Figure 30) to change the LDAP server's configuration.

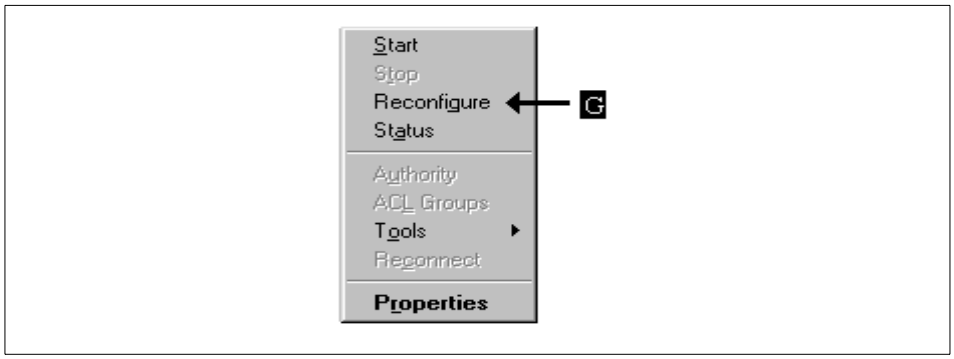


Figure 30. LDAP administration pop-up menu

Configuration vs. Reconfiguration

When you use the wizard to reconfigure the directory server, you actually start from scratch. The original configuration is deleted rather than changed. You can modify the directory server configuration by right-clicking **Directory** and selecting **Properties**. This does not delete the original configuration.

To begin the configuration process using the configuration wizard, click on **Next** (Figure 31).

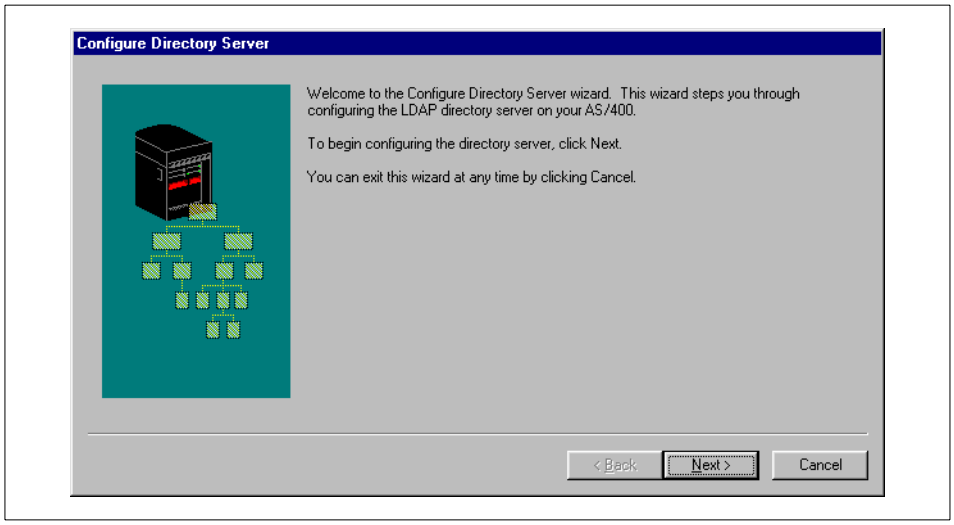


Figure 31. LDAP configuration wizard - Welcome screen

Enter the AS/400 library for the relational database in the field as shown in Figure 32. The library name must be specified using IFS naming. You should specify a library that will only be used by the directory server. Click on **Next** to proceed. If the library you specified does not exist, the wizard will ask you for confirmation to create this library. If you have multiple ASPs (Auxiliary Storage Pools) on your system, you will be prompted to select an ASP.

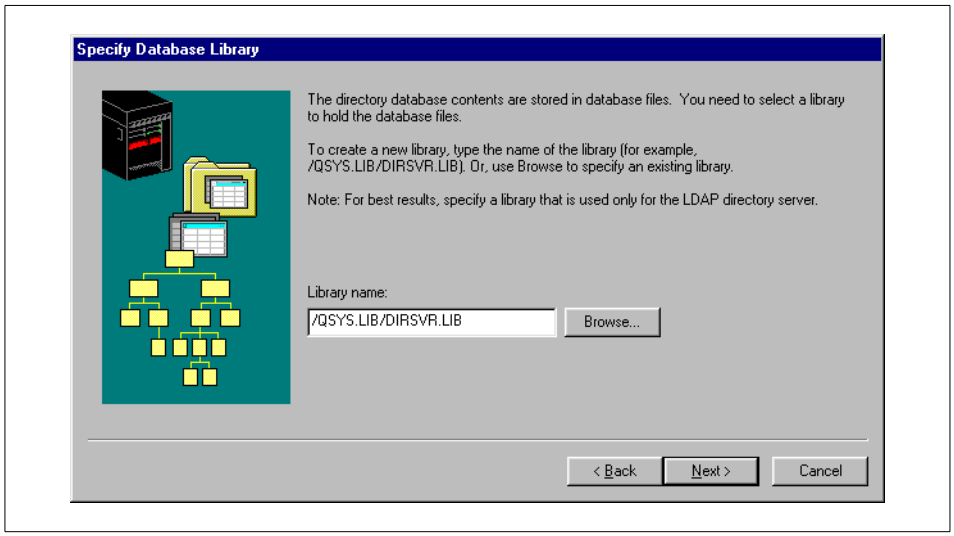


Figure 32. LDAP configuration wizard - Specify database library

Specify the distinguished name (DN) of the administrator and the password for the directory server in the fields shown in Figure 33. The name string uniquely locates the entry within your directory server directory. Type the name and password values and click **Next**.

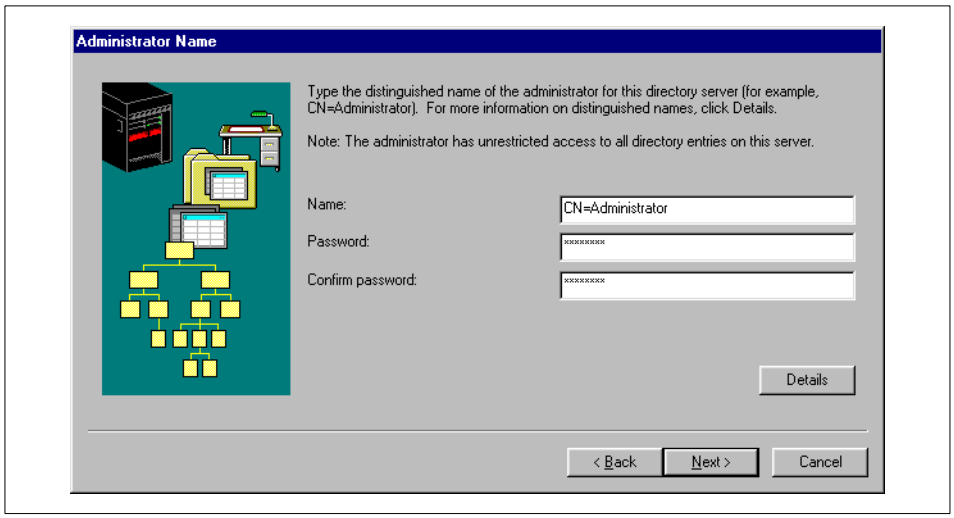


Figure 33. LDAP configuration wizard - Administrator name

Enter the directory suffixes at the next screen; an example is shown in Figure 34. You must configure at least one suffix in addition to the <cn=localhost> (for example, <O=IBM, C=US>). Type the directory suffix, then click **Add** to add the entry to the directory server. After you have added all the entries you need, click **Next** to continue.

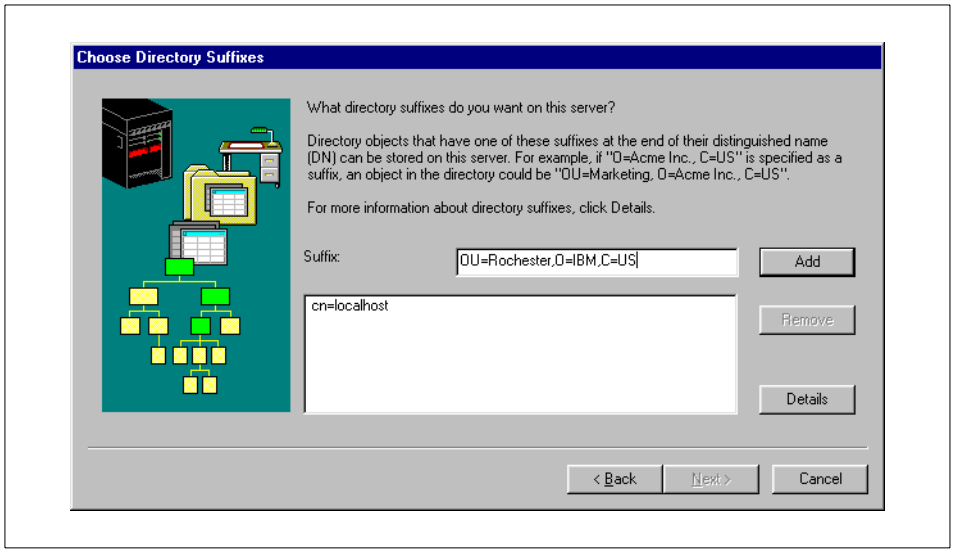


Figure 34. LDAP configuration wizard - Choose directory suffixes

Using the check box shown in Figure 35 as an example, specify if you want the LDAP server to start when TCP/IP starts. Make your selection and click on **Next** to continue.

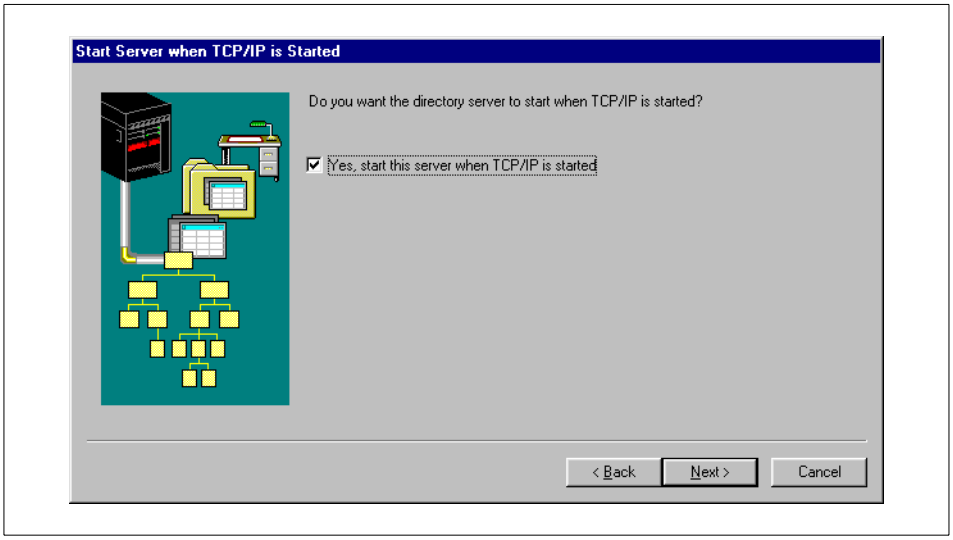


Figure 35. LDAP configuration wizard - Start server option

Review the configuration summary screen shown in Figure 36. Verify the settings for the directory server. If needed, you can go back and correct configuration mistakes by clicking **Back**. When all the information is correct, click **Finish** to complete the configuration.

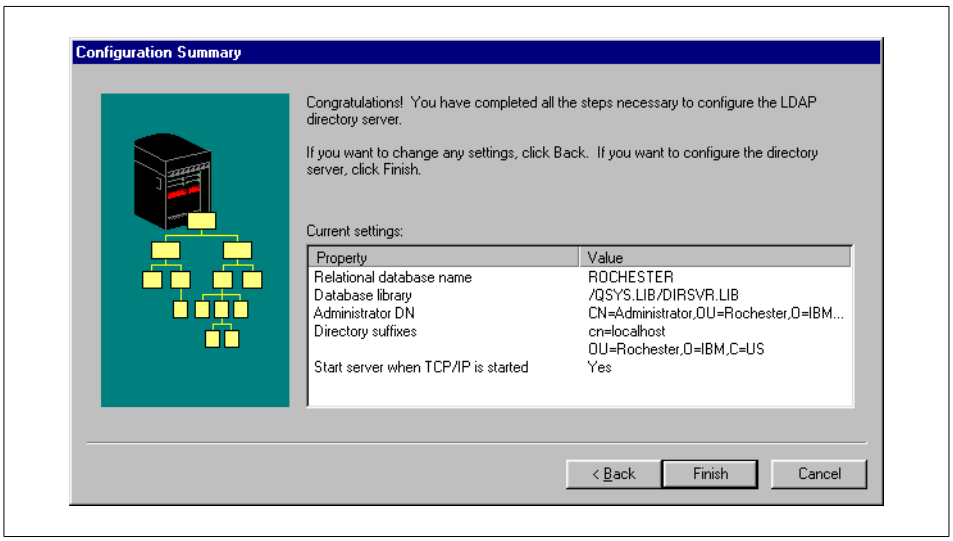


Figure 36. LDAP configuration wizard - Configuration summary

This concludes the configuration of the IBM SecureWay Directory on AS/400.

After you have finished the configuration, you can start and stop the directory server using the following commands. To start the LDAP server, type:

```
STRTCPSVR SERVER(*DIRSRV)
```

Similarly, to stop the LDAP server, type:

```
ENDTCPSVR SERVER(*DIRSRV)
```

During the first start, all needed files are automatically created in the relational database library you specified during configuration.

6.4.4 Uninstalling the Server

If you no longer wish to run an LDAP directory server on your AS/400, you can remove it by uninstalling the AS/400 Directory Services option of OS/400.

To do this, take these steps from a 5250 session to your AS/400:

1. Type GO LICPGM, then press **Enter**.
2. Choose option **12** from the Work with Licensed Programs menu, then press **Enter**.
3. Enter **4** in the Option field to the left of Option 32, OS/400 - Directory Services, then press **Enter**.

After you complete this procedure, the server will be uninstalled. However, both the schema files and the library that contains the server's data remain on the AS/400. This allows you to easily reinstall the AS/400 Directory Services and to begin running the server again.

If you are sure that you will not want to run the LDAP directory server in the future, you may delete these items. The schema files are located at /QIBM/UserData/OS400/DirSrv and their names are UsrAt.txt and UsrOc.txt.

Chapter 7. LDAP Data and System Administration

In Chapter 4, “Managing an LDAP Directory” on page 77, management of an LDAP directory from a high-level planning perspective was explained. This chapter exploits in more detail the administration of an IBM SecureWay Directory server system and related tasks. Administration of the IBM SecureWay Directory is most conveniently done through the provided Web-based Administrator Graphical User Interface (GUI). This chapter also gives an overview of the utilities provided with the IBM SecureWay Directory and the IBM SecureWay Directory Client SDK that can be run from a command prompt to manage product specific data. Finally, the schema file(s) management is also describes in this chapter.

7.1 The Directory Management Tool

The IBM SecureWay Directory Management Tool (DMT) is an easy-to-use, Java-based graphical tool for directory administrators for browsing and checking directory objects and to add and update such objects. It is included in both the IBM SecureWay Directory and the IBM SecureWay Directory Client SDK. The following figure shows the initial DMT window.

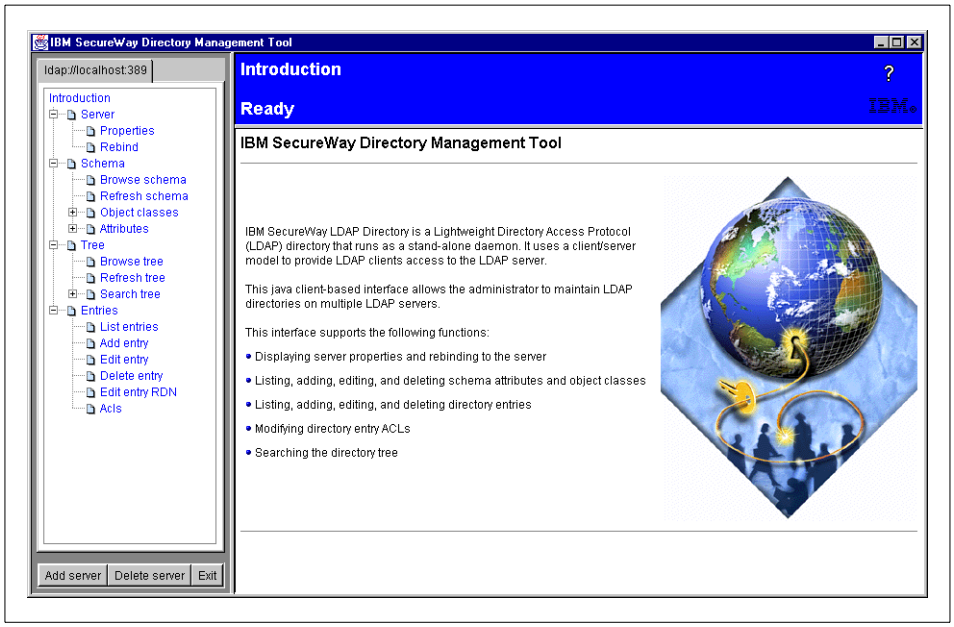


Figure 37. IBM SecureWay directory management tool

7.1.1 Startup and Configuration

The DMT can be started by running the dmt executable program (x:\<ldap_dir>\bin\dmt.exe in Windows NT) or by selecting the **Directory Management Tool** menu item under the IBM SecureWay Directory start menu (Windows NT). The DMT uses a configuration file dmt.conf that stores some basic information. A default, sample dmt.conf file, looks like this:

```
server1.url=ldap://localhost:389
server1.security.bindDN=
server1.security.password=
server1.security.ssl.keyclass=
server1.security.ssl.keyclass.password=
```

As can be seen, the dmt.conf file stores information about an LDAP server to which it connects automatically upon startup. Any server specified in this configuration file will show up as a tab at the top of the left side in the DMT window (see Figure 37). This file can be edited manually, for example, to add additional LDAP servers to which you want DMT to connect automatically in order to manage its contents. The bindDN and password information is optional; if not specified, DMT will bind to the server as *anonymous*, but you can authenticate later through the tool. Additional servers can also be added (or deleted) by clicking the respective button in the lower left corner of the DMT window. This, however, will not update the dmt.conf configuration file.

Note

Specifying a password in the configuration file exposes a significant security risk. It is, therefore, a good practice not to specify a bindDN and password in the configuration file and bind to the server through the GUI instead.

Bear in mind, unless you authenticate as a user with appropriate privileges (for example, as a directory administrator), that you have only anonymous privileges when using the DMT. To authenticate, select **Server**, then **Rebind** from the menu tree on the left-hand side. You will be prompted for a user DN and a password.

The use of the DMT is rather intuitive and does not need much explanation. We recommend that you use the DMT to familiarize yourself with the directory structure (even if you have only the sample directory imported from the sample.ldif file installed) and the many object classes and attributes defined by default that are not mentioned elsewhere in this book. The example in the following section illustrates the ease of use of the DMT.

7.1.2 Example: Expanding the Schema

The following example adds two attributes to the ePerson object class: A job description (jobDesc) and a job level (jobLevel) attribute. The former shall be a case-insensitive string, the latter an integer. Such an example has been discussed on page 32, and two possible solutions have been suggested. We choose the second proposal by adding a separate auxiliary class (myPerson) that contains the additional attributes (jobDesc and jobLevel). The following shows what you have to do in order to accomplish this with the DMT.

First of all, you need to add the two attributes jobDesc and jobLevel. In the DMT window, expand the Schema and Attributes menu tree on the left hand side, then click on **Add attribute**. In the dialog that appears, fill in the information for jobDesc as shown in Figure 38.

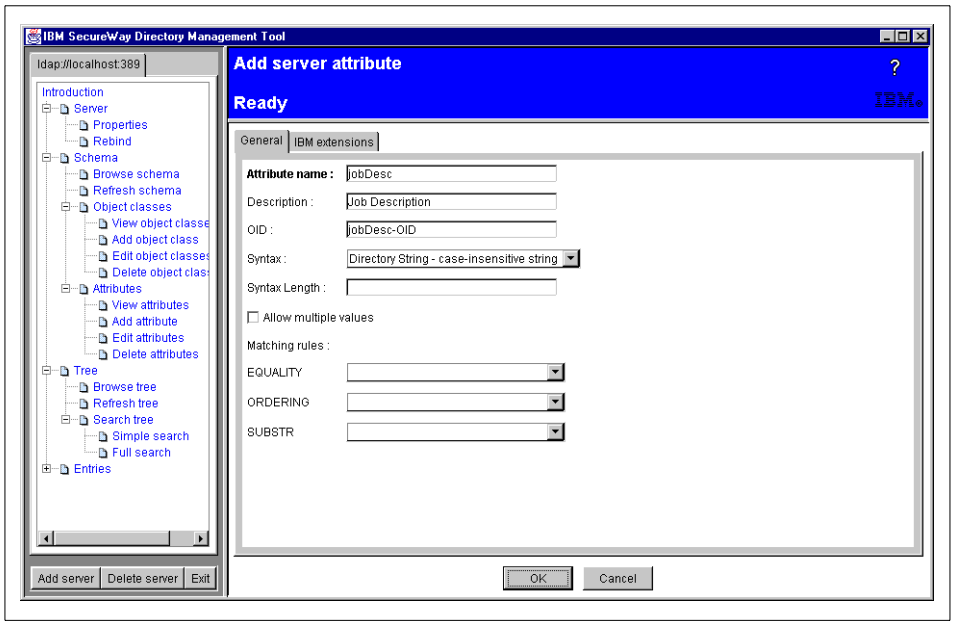


Figure 38. Adding an attribute

Optionally, you could define matching rules and properties for the DB2 database. We keep it simple and just define the attribute's name and its syntax to be a case-insensitive string.

After this, you would do the same for the jobLevel attribute where INTEGER would need to be selected as the syntax.

Next, a new auxiliary object class, myPerson, needs to be added. Select **Schema** -> **Object classes** -> **Add object class** from the menu tree on the left. The dialog shown in Figure 39 shows that you are allowed to enter a name (myPerson), a description, and a superior object class. Then, click on the **Optional attributes** tab at the top and add jobDesc and jobLevel to the list on the right (Figure 40). Then click **OK** to add the object class.

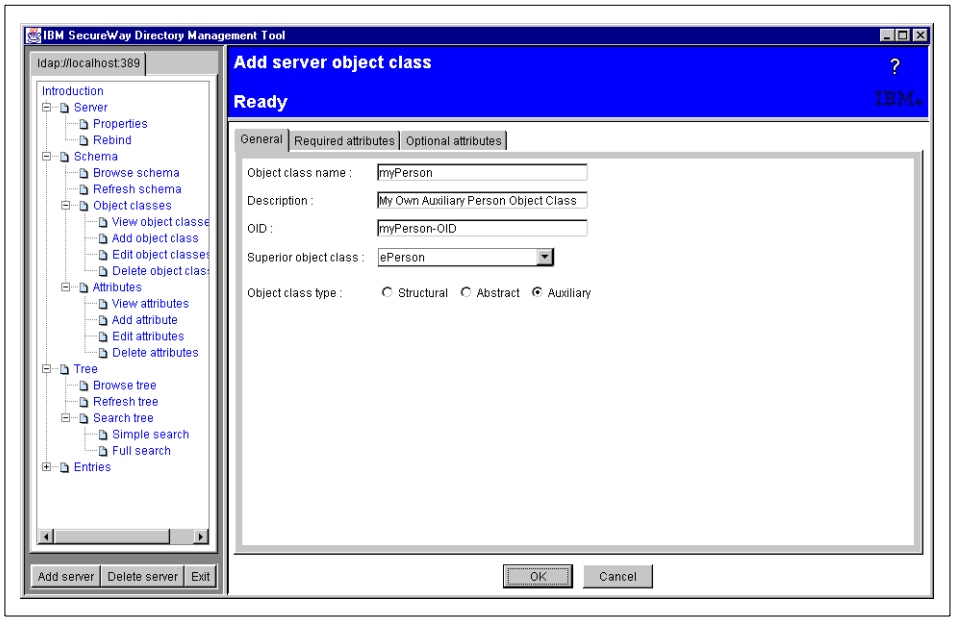


Figure 39. Define a new object class - Name and superior class

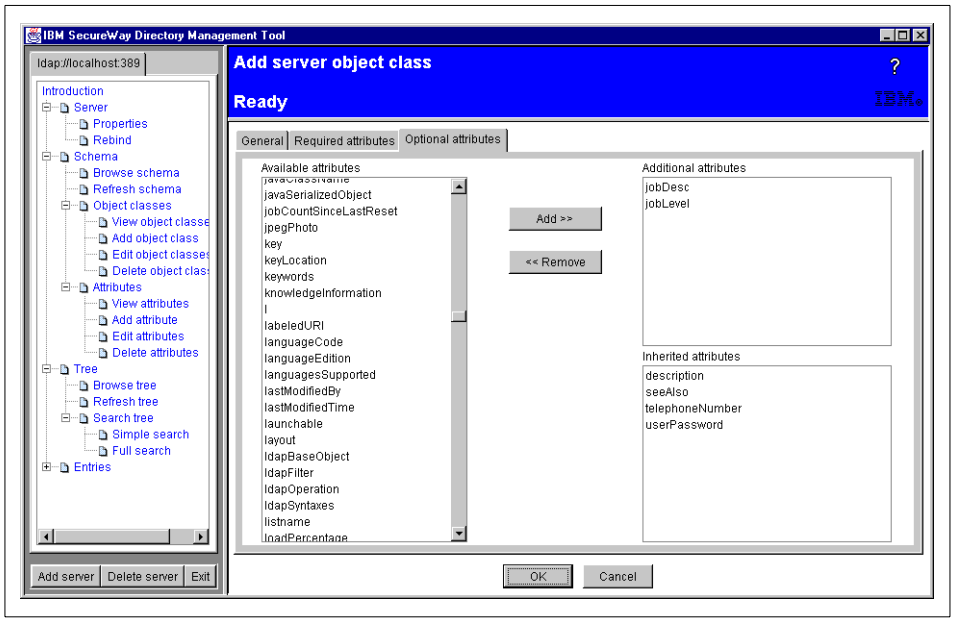


Figure 40. Define a new object class - Optional attributes

The new object class has now been added, and you can proceed and add objects (people entries) to the directory using this object class. Select **Tree** -> **Browse tree** in the menu tree on the left and then expand the directory tree as necessary to select (highlight by single-click) the entry to which you want to add a person (in our example, <ou=ITSO,ou=Austin,o=ibm,c=us> has been chosen). Then, click the **Add** button at the top to add a new entry to launch the following dialog (Figure 41):

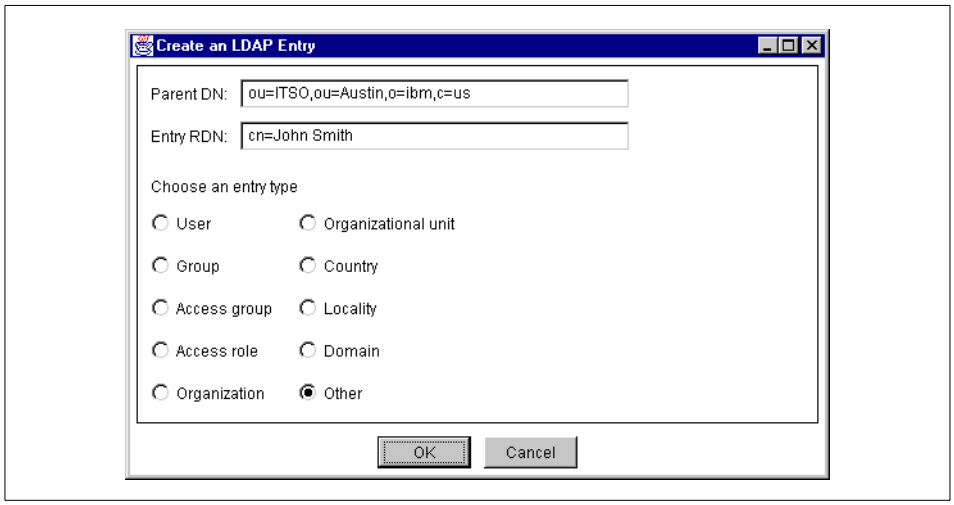


Figure 41. Add a new person using the new object class

The new person in our example is *John Smith* and the **Other** entry type has to be selected (Figure 41) because we use a non-standard entry type. Click on **OK** to proceed to the next dialog that requires you to select the object class(es) for this entry (Figure 42). We need to select the **`person`** structural object class and then the **`ePerson`** and the new **`myPerson`** in the auxiliary object classes list underneath (notice that this is a multi-select list). The Current definition tab shows you all the attributes for the object for information and verification. Click **OK** to move on to the attributes dialog where the attribute values can be defined (Figure 43).

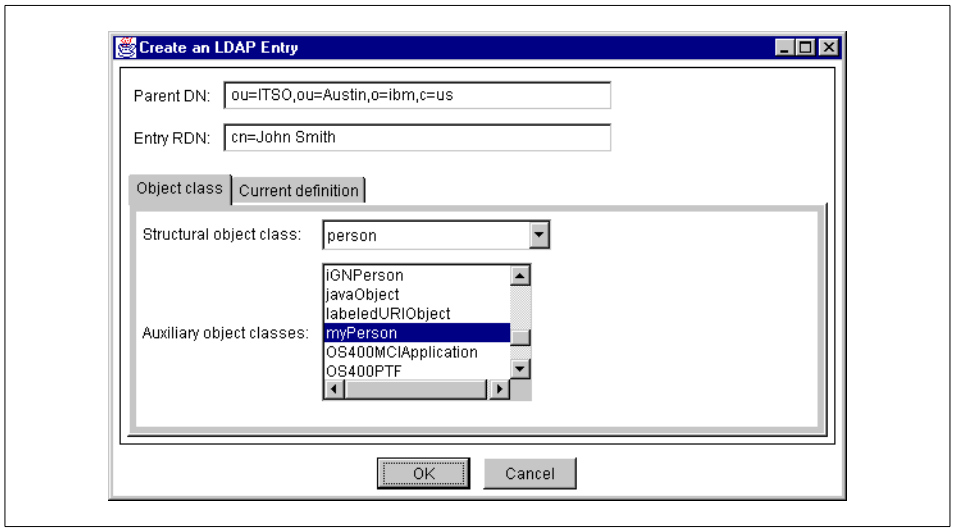


Figure 42. Select the object classes for the new entry

Make sure that the required attributes (cn and sn, inherited from the person object class) are filled in and scroll down the list to see the new attributes jobDesc and jobLevel (Figure 43). Click **Create** to create (add) the new entry.

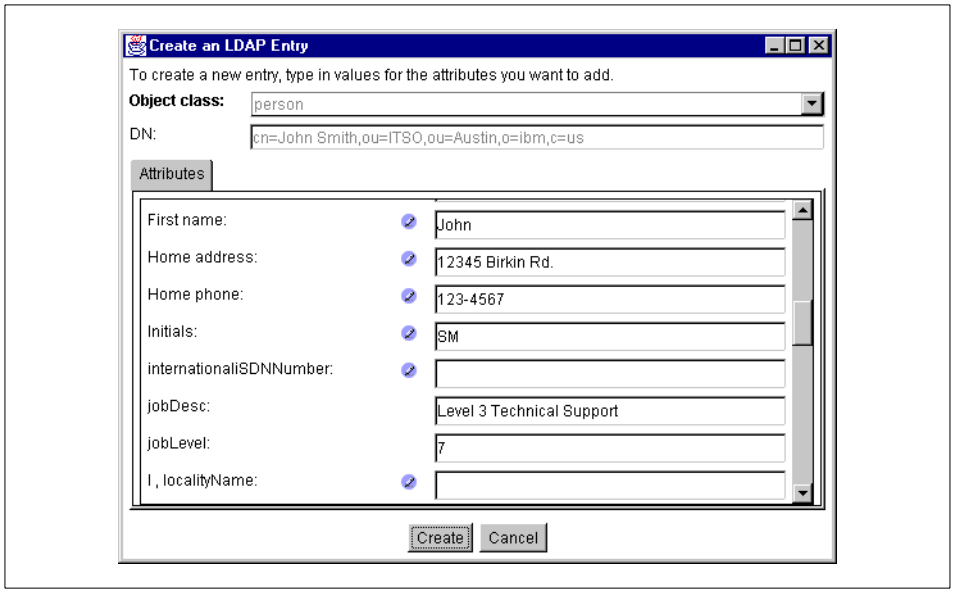


Figure 43. Enter the attribute values for the new entry

After this entry has been added, it shows up in the Browse tree view of the directory as shown highlighted in the following figure.

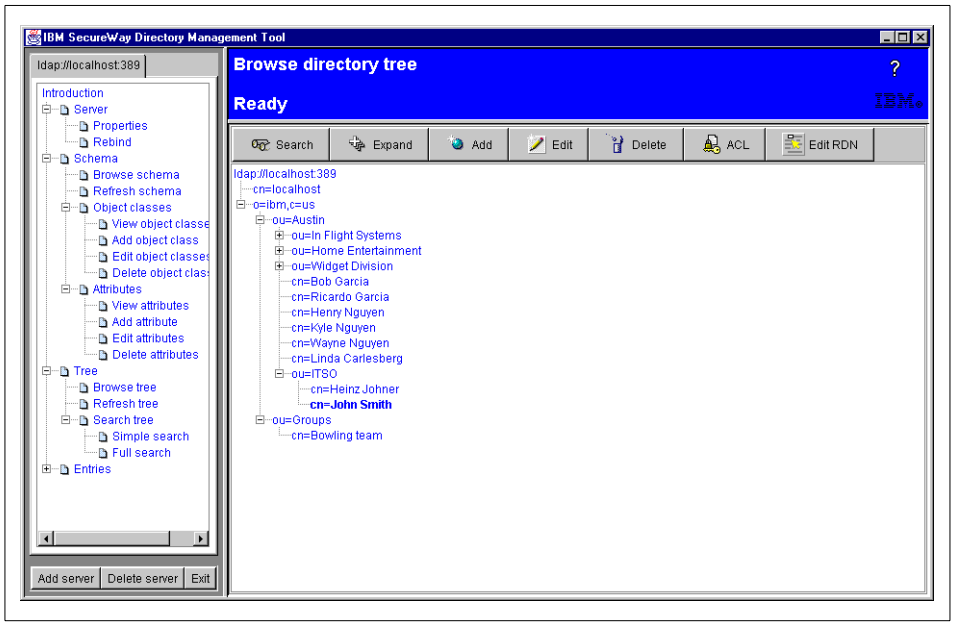


Figure 44. New person entry in the directory tree

This simple example shows how easily the directory schema can be dynamically modified and applied to entries. It is, however, recommended to use the standard schema whenever possible to avoid possible complications when moving contents to or from other directories or when upgrading to future releases as discussed in Chapter 2, “Schema and Namespace” on page 31.

7.2 The Administrator Graphical User Interface

The Web-based Administrator Graphical User Interface (administration GUI) is provided with the IBM SecureWay Directory V3.1 (on AIX and Windows NT at the time of writing) to assist the administrator. This interface allows the administrator to configure the directory server and the underlaying database and to administer the tasks associated with maintaining the database. The procedures for tasks are displayed step-by-step. Using the administration GUI ensures that no mistakes are made during configuration and management tasks by verifying that the correct options are selected and are then formatted correctly. You can access the administration GUI using an

HTTP connection from a Web browser on any system to the Web server. This interface allows the administrator to set up and maintain a database/directory LDAP server from virtually any system on the network.

7.2.1 Launching the Administrator GUI

To get the administrator GUI, use an HTTP connection from a Web browser. Type in the following Web address:
`http://<web_server>:<port>/ldap`

Where:

`<web_server>` is the name or IP address of your Web (LDAP) server, and `<port>` is the port your server is using.

You are then prompted for the administrator's distinguished name (DN) and password. Use the same DN and password you defined when you configured the LDAP server (see Chapter 6, "Installation and Configuration" on page 131), for example <cn=root,ou=ibm_us,c=us>. If you do not remember the full name, you can find the `aminDN` at the top of the slapd.conf file located in <ldap directory>/etc. For example, you can find the following line:

```
adminDN    "cn=root,ou=ibm_us,c=us"
```

If login was successful, the administrator GUI opens in your browser.

7.2.2 Window Layout

The administration GUI's general window layout (see Figure 45) consists of three areas, which are:

- The assistance area
- The navigation frame
- The work area

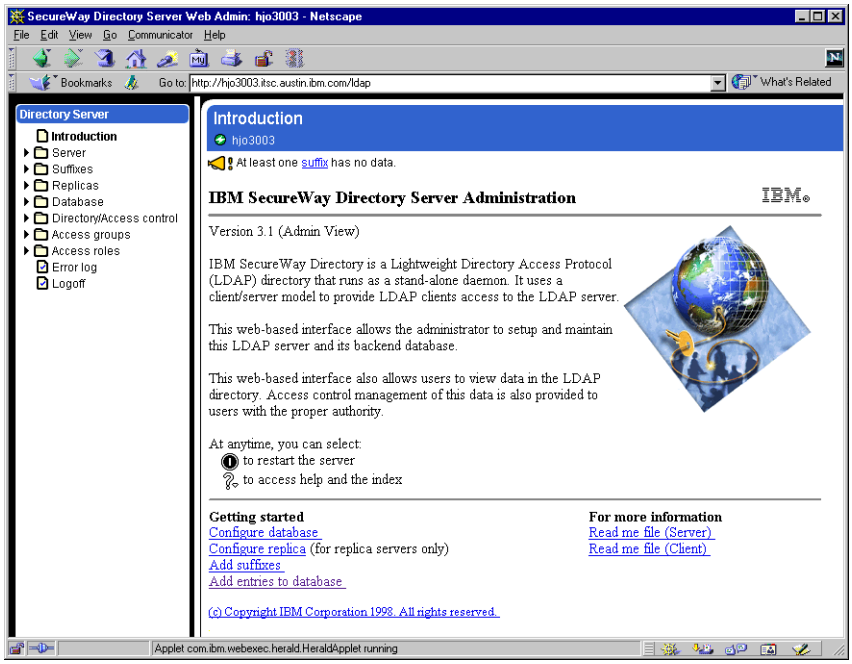


Figure 45. Initial administration interface

The Navigation frame is the left frame in the window. It contains all objects that you can select to work with the directory. They are:

- **Introduction** – Displays the welcome screen
- **Server** – Work with general server settings
- **Suffixes** – Work with (list, add, delete) the server's suffixes
- **Replicas** – Work with replication setup
- **Database** – Work with database properties
- **Directory/Access control** – Browse the DIT, work with DNs
- **Access groups** – Work with access groups
- **Access roles** – Work with access roles
- **Error log** – Browse, clear the server's error log
- **Logoff** – Logoff from the administration GUI

Only tasks available for the server are shown. If you are on a replica server, for example, and you want to change access control for one object, the task will not be shown because you must be on a master server to perform this task. When you click on an object in the Navigation frame, the task options are displayed in the work area to the right. Note that the small triangles (also

called *twisties*) on the left denote that this object has sub-objects that will be shown as you click on the particular object.

The Assistance area is located on the top of the right frame of the window. From this area, you can restart the server and get online help. It also displays some status information, such as on which server you are working and what you are currently doing, as for example, *Working with server suffixes*.

The Work area is the large area to the right of the Navigation frame. This is where interaction occurs for the various functions.

7.3 Database Configuration

The first step you must perform as a directory administrator after installing the IBM SecureWay Directory is to configure the database. Note that this might have already been done during installation and basic configuration.

Your namespace and directory structure are supposed to be defined with the type of data you are going to store in the directory. If you want to exercise the LDAP administration without your own data, you can use the sample file provided with the LDAP package. The filename is sample.ldif, and it can be found in the examples subdirectory underneath the LDAP installation directory (for example, C:\Program Files\IBM\LDAP on Windows NT).

Before configuring the database, you have to decide if you want to use your own database or the default one proposed by the product. For test environments and many installations of the LDAP directory, the default setup will be adequate. However, some specific installation may need or want to rearrange the data for performance and/or extensibility reasons. If you are in doubt, we recommend that you study the White Paper that is available to get a better understanding about how the LDAP directory is implemented and how the LDAP directory tablespace is organized along with other pertinent information. You can get this White Paper from the Web site:

`http://www.ibm.com/software/network/directory/library/whitepapers/ldap_scaling.html`

To configure the database, click on **Configure database** in the Introduction work area to get the following window (in Figure 46, only the relevant portion of the work area is shown). Select the type of database you want to use, then click on the **Next** button.

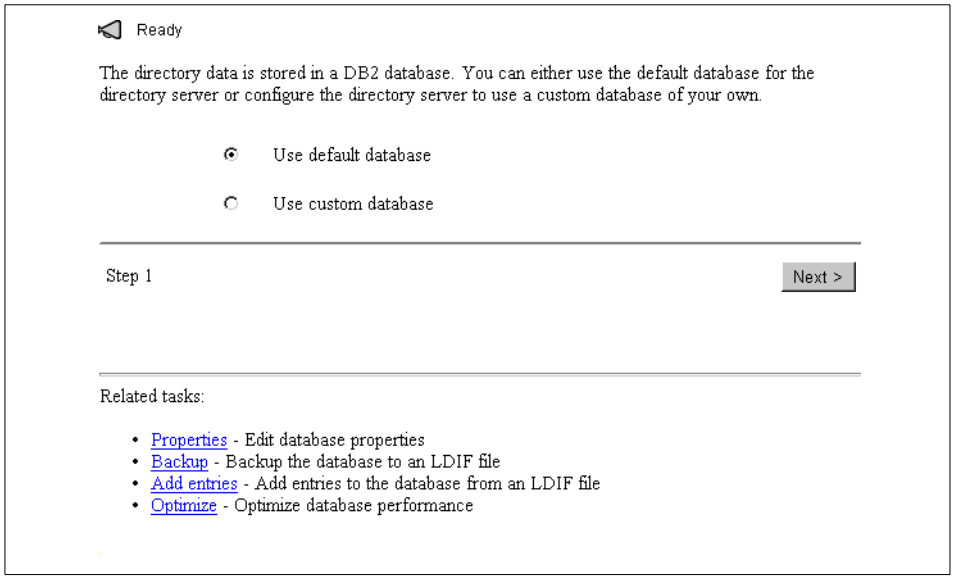


Figure 46. Configure database

The following two sections explain the two options.

7.3.1 Default Database

The default name for the database is ldapdb2; the DB2 instance owner created is also ldapdb2.

Click on the **Use default database** radio button, then click on **Next>** to proceed to the next panel. If a database already existed on that system, you will be given the chance to enter a file name for a backup file (in LDIF format) before the database is erased and recreated. On the next panel, define the DB2 location.

- Choose the hard disk location for Windows NT.
- For AIX, the default directory is /home/ldapdb2, which cannot be changed. If your database resides at another location (other than /home/ldapdb2), you should add a link to that directory. For example, if your database is at /ldapdb2, issue the following command:

  ```
  # ln -s /ldapdb2 /home/ldapdb2
  ```

- Click on **Finish** to start the creation of the database.

Note

It is recommended to back up any current database that might have existed before creating the new one. The current database will be deleted.

7.3.2 Custom Database

If you want to use your own database, the minimum size should be approximately 80 Mb. Select **Use custom database** (Figure 46) and click on **Next>**, then follow the instructions (optionally, backup an existing database). The following information is needed for setting up the database:

- Database name
- Database instance
- Database system administrator ID
- Database system administrator password

Note

The database name ldapdb2 is reserved for the IBM SecureWay Directory; do not use it for your own database name.

Click on **Finish** to start the configuration.

DB2 Scalability and Performance

DB2 is a powerful and flexible product. An experienced administrator may want to tailor its performance and scalability characteristics to a specific environment. To do that, DB2 offers many methods and techniques to optimize database performance after significant database updates have been done or as a regular maintenance practice.

DB2 database management is beyond the scope of this book, but you can find information on tuning DB2 and how the LDAP directory utilizes DB2 as the directory storage facility on the following Web pages:

`http://www.ibm.com/software/data/db2/library`

and

`http://www.ibm.com/software/network/directory/library/whitepapers/ldap_scaling.html`

7.4 Defining a Suffix

Before entries can be added to an LDAP directory, a *suffix* must be defined for that directory. A suffix specifies the distinguished name (DN) for the root of

a directory in the local database. It is the highest entry stored in the directory by a server. Each entry stored by the server ends with this suffix, and each entry to be added to the directory must have a suffix that matches one of the server's suffix defined on server. An LDAP server can house multiple suffixes, and, thus, you can add and delete suffixes to or from the directory:

To add a suffix to the LDAP directory, perform the following:

1. Click on **Suffixes** in the Navigation frame
2. Click on **Add a suffix** in the Work with servers suffixes work area (or in the Navigation frame if you clicked on the *twistie* next to Suffixes)
3. Type the suffix DN in the text entry field, for example: `o=ibm_uk,c=uk`
4. Click on **Add a new suffix**.
5. Restart the server.

To delete a suffix from a directory:

1. Click on **Delete suffixes**.
2. Click on the box next to the suffix you want to delete.
3. Click on the **Delete suffixes** button.
4. Restart the server.

Note

Removing a suffix will disable access to all directory data underneath that suffix. This does not physically remove the data from the directory. Access can be restored to a deleted suffix by adding it again. The suffixes are recorded in the slapd.conf file (though it is not recommended to manually edit this file).

7.5 Database Population

To add entries to the directory, you can either use the DMT (see 7.1, "The Directory Management Tool" on page 159) or you must provide the data in a file in LDIF format. LDIF (LDAP Data Interchange Format) is a standard format for representing LDAP entries in text form (see also 4.7.1, "The LDIF File Format" on page 93). Also, a suffix (DN) must exist on the server (see previous section), and all entries to be added must have a suffix that matches this DN value.

7.5.1 Adding Data Entries

There are two ways to import an LDIF file, either through the graphical user interface (GUI) or by using the appropriate command on the command line. Both methods are described below.

7.5.1.1 Using the Graphical User Interface (GUI)

For the example below, the sample LDIF file shipped with the product was used. Some editing is usually necessary to adapt it to the test environment, for example, to define the correct suffix. In addition, an error was introduced on purpose in the LDIF file to show the server's behavior.

1. Click on the **twistie** next to Database in the Navigation frame and then on **Add entries** in either the Navigation frame or the Working with the backend database work area.
2. Specify an LDIF file. In this example, the sample.ldif file provided with the product was used.

   <ldap root directory>/examples/sample.ldif

   This file has been modified manually for demonstration purposes: <o=IBM_US> was changed to <o=ibm_uk>, <c=US> was replaced by <c=uk>, and <ou=Austin> was changed to <ou=nhb>. The DN for Michael Lord was changed to contain an error (<o=ibm_nl, c=nl>, which is an invalid suffix). This is an excerpt from the altered LDIF file:

```
dn: o=ibm_uk, c=uk
objectclass: top
objectclass: organization
o: IBM_US

dn: ou=nhb, o=ibm_uk, c=uk
ou: Austin
objectclass: organizationalUnit
seealso: cn=Linda Carlesberg, ou=nhb, o=ibm_uk, c=uk

dn: ou=In Flight Systems, ou=nhb, o=ibm_uk, c=uk
ou: In Flight Systems
objectclass: organizationalUnit
description: main product:Course Maker
businessCategory: aircraft
seealso: cn=Maria Garcia, ou=In Flight Systems, ou=nhb, o=ibm_uk, c=uk

dn: ou=Home Entertainment, ou=nhb, o=ibm_uk, c=uk
ou: Home Entertainment
objectclass: organizationalUnit
description: main product:TV Connection
```

```
businessCategory: Home Entertainment

dn: cn=Michael Lord, ou=Widget Division, ou=nhb, o=ibm_nl, c=nl
objectclass: organizationalPerson
cn: Michael Lord
sn: Lord
telephonenumber: 00-44-999-5838
internationaliSDNNumber: 999-5838
postalcode: 4681
```

3. Enter the location of the file containing the entries in the text entry field and click on **Add entries to database**. This loads the LDIF file into the directory. If the import was not successful, an error indication is shown in the status line at the top (in Figure 47, only a portion of the window is shown).

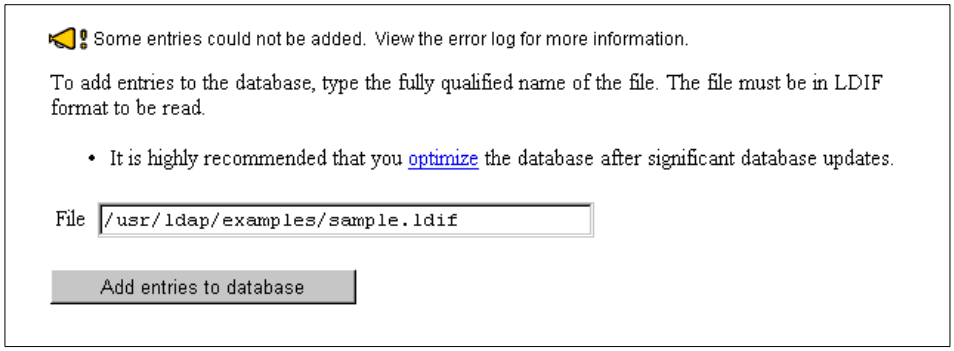


Figure 47. Importing an LDIF file (with errors)

4. If the import was not successful, such as in this case, click on **Error logs** in the Navigation frame. An error message, such as the one in Figure 48, will be shown.

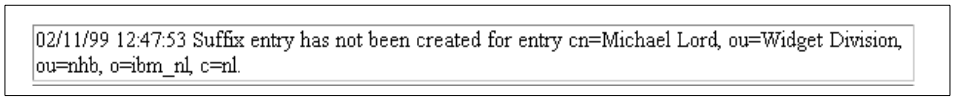


Figure 48. Import error message

In this case, the error was caused because the suffix defined in the directory (and the LDIF file) is <o=ibm_uk,c=uk>, but the entry for <cn=Michael Lord> contains <o=ibm_nl,c=nl>.

7.5.1.2 Using the Command Line Tool

As an alternative to the GUI, the `ldif2db` utility can be used to load entries from an LDIF file into a directory. The database and suffix must exist, and the file must be in LDIF format. The syntax is:

```
ldif2db -i <input file>
```

The `ldif2db` utility may be used to populate an empty directory or to load data into a directory that already contains entries. You should always check the Error log from the GUI (or by browsing the error file defined in the slapd.conf configuration file). If you want to change the location of this file, click on **Server** in the Navigation frame, then **Properties** in the Working with the LDAP server work area, click on **logging** in the Server properties work area and change the Error log path. The server needs to be restarted for this change to take effect.

More about the ldif2db utility can be found in 7.8.2, “The LDIF2DB Utility” on page 189.

7.5.2 Verifying Data Entries

Scanning the error log is a good way to verify whether or not an update has been successful. Remember that an LDIF file may contain a large number of entries, and errors may occur on just single entries of such a bulk load or update. For this reason, you should always check the error log. Another way to check the success of an import, though not suitable for a large number of entries, is to list the contents of the directory to verify if an entry is or is not present. You can use the DMT, the administration GUI, or a command line utility for checking entries.

7.5.2.1 Using the DMT

Select the **Tree** -> **Browse tree** function from the menu tree on the left-hand side of the DMT window (see Figure 37) and then work your way down the directory tree in the work area to verify whether a certain entry (or entries) is correctly imported.

7.5.2.2 Using the GUI

1. Click on **Directories/Access control** in the Navigation frame.
2. Click on **Browse tree** in the Working with the entries in the directory work area.
3. Click on the **check box** next to the suffix you want to display.
4. Click on the **check box** next to the top directory you want to display. For example, if the directory was populated with the LDIF file of the example

above and you went down through the branch <ou=nhb>, you could not find an entry for Michael Lord because it was refused due to an error.

The update was successful when all entries required to be in the directory are effectively listed by the tool.

7.5.2.3 Using the Command Line Tool

The ldapsearch (see also 7.8.5, “The LDAPSEARCH Utility” on page 192) utility can be used to check certain entries in the directory. For example, this could be done with the following command:

```
# ldapsearch -b "o=ibm_us,c=us" cn=*
```

This command will perform a search of all common names (cn) defined under the suffix <o=ibm_us,c=us>.

7.5.3 Updating Data Entries

Directory entries can be updated one-by-one with the DMT or multiple at a time by importing an LDIF file with appropriate update statements either by using the GUI or using the command line tool as described below. Remember that an LDIF file can only contain new entries or updates to current entries; it cannot contain both in the same file (see also 4.7.1, “The LDIF File Format” on page 93).

7.5.3.1 Using the GUI

Follow these steps to load an LDIF file with updates in it:

1. Create an LDIF file containing the updates for the entries.
2. Click on the **Database** twistie in the Navigation frame and then on **Add entries**.
3. Enter the full path of the LDIF file and click **Add entries to database**. Watch for any error indication after the file has been imported.
4. Optionally, you may use the Browse tree function to check if the new entries have successfully been changed.
5. Click the **optimize** link in the Working with the backend database work area to perform a database optimization (only recommended when major changes have been made). Be aware that, during this reorganization, the database is not available, and the LDAP server needs to be restarted afterwards.

Note

It is recommended that you optimize the database after significant database updates.

7.5.3.2 Using the Command Line Tool

The `ldif2db` tool can be used to load entries through an LDIF file into a directory. The syntax is:

```
ldif2db -i <input file>
```

You can check the Error log from the GUI or by browsing the error log file as defined in the configuration file slapd.conf. More about the ldif2db tool can be found in 7.8.2, “The LDIF2DB Utility” on page 189.

7.5.4 Back up the Database

You can back up the database either using the GUI or using a command line tool. The steps are described below.

7.5.4.1 Using the GUI

1. Click on **Database** in the Navigation frame.
2. Click on **Backup** in the Working with the backend database in the work area.
3. Type the fully qualified name of the file to be created in the text entry field. The file will be in LDIF format.
4. Click on the **Backup database** button to start backing up the database.

7.5.4.2 Using the Command Line Tool

The `db2ldif` tool (see also 7.8.4, “The DB2LDIF Utility” on page 192) can be used to dump entries from a directory to a text file in LDIF format. The syntax is:

```
db2ldif -o <output file> [-s <subtree>]
```

Be aware that the LDIF format does not honor any ACL settings, and that all data is available in readable text.

Using NativeDB2 Backup

The alternative solution for backup is to use native DB2 utilities. To do that, you need to be authenticated to the database. Please refer to the DB2 documentation for more information about database backup and restore.

7.6 Replication

Replication can be used to improve performance by providing a service from multiple machines in order to satisfy a search as quickly as possible. Replication can also be used to improve the availability of a directory service by having more than one server. If one of them is temporarily down, the directory service continues to be available from another server. The IBM SecureWay Directory replication is based on a master-slave replication model.

There are two types of directories with respect to replication: *Master* and *replica*. LDAP refers to the master as *master server* and to the replica as *replica server.* For a particular directory structure, there can only be one master server; all updates are made on the master server, and these updates are subsequently propagated to the replica server(s). Every replica server's database contains an exact copy of the master server's directory data. A replica server can be promoted as master server if required (for example, if the master server is out of service for an extended period of time) in order to allow write operations to the directory during this time.

Changes can only be made to the master server. If a replica server receives a request to update an entry, this request will be returned with a reference to the master server using a referral (see the following section 7.7, "Referrals" on page 184).

If you remove a replica server, an action must be done on the master server to remove the replica server's definition. Without that, the master server will continue trying to update the missing replica server.

7.6.1 Configuration

There are two types of configuration. The first is when the database of the master server is not yet populated, and the second is when you are to create a new replica server for a directory that is already loaded with data. Once the configuration is done, you have to set up the replication process. Like configuring replication, you have two possible scenarios: A simple setup for simple configuration, and a general setup when the master server has already been populated with data.

Because replica servers handle the same directory context, you must make sure that all replicas defined in the directory context have the same suffixes as the master server, and also the schema definitions must be the same on each replica and the master server.

In the following, the two replication setups, as mentioned above, are explained.

7.6.1.1 Simple Configuration

This method is to be used when the master server does not contain any data. After basic configuration of each individual server, one or more of them need to be configured as replica servers. The replica servers must be known by the master server; once the directory data is loaded to the master server, the replica servers can be started, and then the master server propagates the data.

The steps are the following (assuming a master and one replica server):

1. At first, install and configure both servers, master and replica, without regarding which one is going to be the master or replica.
2. Configure a database on each of them according to the description in 7.3, “Database Configuration” on page 169.
3. The configuration of the replica server must be done before the initialization of the master server. After basic configuration, each server is, by default, a master server. Thus, the designated replica servers must be configured to become replica servers:
   - Click on the **twistie** next to Server in the Navigation frame of the GUI, then click on **Master/replica configuration**.
   - Type the master DN (for example cn=replica1) and the password (twice for verification) in the text entry fields.

     The master DN is used by the replicas to communicate with the master server. It can be any name, but this name must be the same on all replicas in the directory context, and it must be the same as defined on master server (see below).
   - Type the master server’s hostname that replicas return to the client for an update request on it in the Referral text entry field.
   - Click on the **Apply** button to initiate the configuration change.
   - Click on **Suffixes** in the Navigation frame and then on **Add a suffix**. Add the suffix as required, but do not restart the server at this time (as suggested by the GUI) because the master server needs to be configured before starting the replica with suffixes information.
4. The master server can now be configured.
   - Click on **Replicas** in the Navigation frame of the GUI, then click on **Add a replica**.

- Enter a common name for the replica in the 'cn=' text entry field. This name must be unique in the directory context. For example, you could use the hostname of a replica server.
- Enter the hostname of the replica server (fully qualified domain name if the domain is not the same as master server) in the host name text field.
- Verify the port number; it should be 389 for unsecured connections or 636 if using SSL. Also, if SSL is being used, set the encryption setting to YES.
- Select the update interval *immediate*.
- Type the master DN (for example cn=replica1) and the password (with verification) as defined in the replica configuration (see above) in the text entry field.
- Click on **Add replica** button to initiate the configuration change.

The master server is now defined and has the knowledge of one or more replica server(s). The current list of replicas known by the master server can be checked as show at the next step.

5. Check the list of replica (optional): Click on **Replicas** in the Navigation frame and then on **List replicas**. You get a panel with the information about replicas previously set up as shown below in Figure 49.

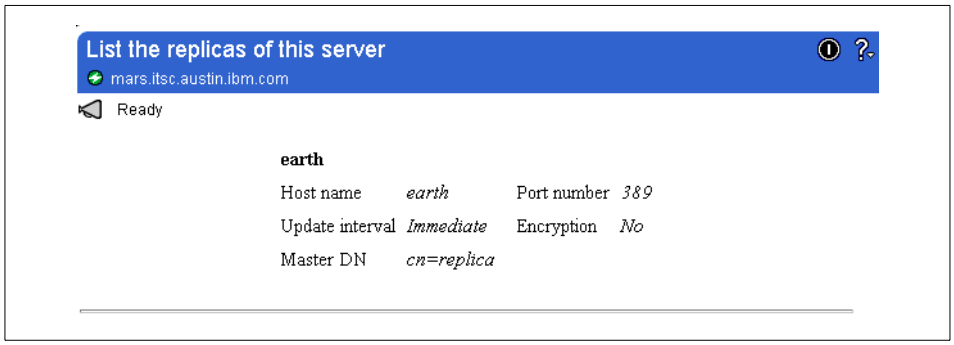


Figure 49. List of replicas

6. Initial replication setup for simple configuration: You have now configured a master server and a replica server in your directory context. Next, you have to ensure that all data that is now imported to the master server is properly replicated to the replica server.

- Load the initial directory data to the master server using either the GUI or the command line utility `ldif2db`, but do not restart the master server at this time.
- Start the replica server.
- Start the master server.
- The master server will now propagate the data that has been loaded to the replica server. The data should be available on the replica server in a short while.

Subsequently, if an update is made to the master server, the change will be propagated automatically to the replica server.

7.6.1.2 General Configuration

The general configuration is used when you need to add a replica to an existing directory context with a master server already operational with data loaded. This general configuration can be used also to reload data onto a replica that was out of service for some time.

The general process is to back up the data on the master server and to import that data in the replica server. Most important is to ensure that the data loaded into the replica is an exact copy of the operational directory data on the master server. The following steps explain the process:

1. Configure the replica server as normal. Do the basic configuration and configure the database as described earlier. You can also add the suffix as defined on the master server that you want to replicate, but do not start the replica server at this time.
2. Configure the master server. Define the replica on your master server, as described for simple configuration above, and do not forget to enter the master DN (and the password) identically on both servers. This step will cause the master server to begin queuing any updates for replication for any replica servers.
3. Stop any updates on the master server. Select the **Read only** permissions in the database properties work area, then click **Apply**. It is not necessary to restart the server as suggested by the GUI.
4. Back up the database. Click on the **twistie** next to Database, and then click on **Backup**. You will then have to enter a name for the LDIF output file. Alternatively, the command `db2ldif -o <output file>` could be used for the same purpose. This creates an LDIF file that you need to import on the replica server in the following step.

5. Using the file created in the last step, import the data to the replica server using either the GUI or the `ldif2db -i <input file>` or `bulkload` commands. The `bulkload` utility is typically used for large databases (see 7.8.3, “The BULKLOAD Utility” on page 189).
6. Start the replica server with suffixes defined in accordance with those defined on the master server.
7. Start the master server (if not already running) or select **Read / Write** permissions in the database properties and restart the server.

This creates a replication setup where the replica server is initially synchronized with the master, and all new updates done on master will automatically be propagated to the replica server.

7.6.2 Promote a Replica as Master

A replica server can be promoted as a new master server. This can become necessary if the current master server is out of service (scheduled or unscheduled) for an extended period of time, while the directory service must remain available for updates.

The following are the configuration steps necessary if you are going to promote a replica as master server.

1. The current master server must be offline (at least the IBM SecureWay Directory server must be stopped). If necessary, click on the **twistie** next to Server, then click on **Startup/Shutdown** to get to the Shutdown button to turn the directory server off.
2. On the replica server, click on **Server** in the Navigation frame, then click on **Master/replica configuration**. The following panel appears (Figure 50).

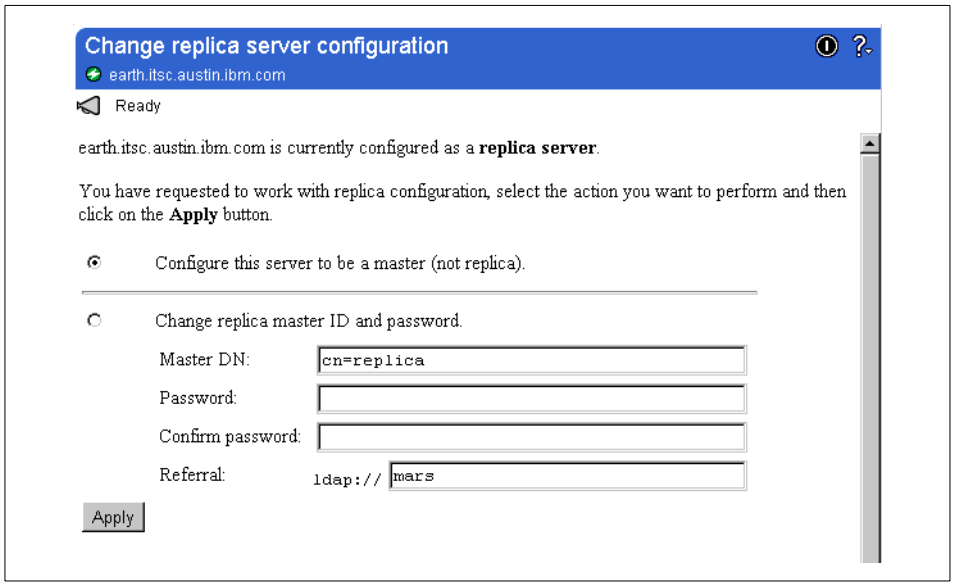


Figure 50. Promote a replica server as master server

3. Select the radio button to the left of Configure this server to be a master (not replica).
4. Click on **Apply** to initiate the configuration change.

This server is now configured as a master server.

The description above promoted a replica server to a new master server while the original master server was offline. At some time, the original master server may be ready to be put back to service. There are two possible scenarios when the initial master server is to be put back into service.

- Keep the newly configured master server (that is, the former replica server) as the master server, and configure the original master server as a replica server.
- Stop the newly elected master server and reconfigure it as it was before, and keep the initial master server as the master server.

In both cases, you have to take care of the updates made to the directory on the temporary master server during this time. All changes made on it will not be forwarded automatically to the server that has later been configured as replica server. You must follow the general configuration and general setup, as described above, to get the directories synchronized. In other words, you

will need to import a full backup of the master server to assure identical directory data.

7.7 Referrals

Referrals provide a way of splitting a namespace into separate partitions as introduced in 1.1.3, “Distributed Directories” on page 6. Scalability, availability, and manageability of an LDAP environment may be improved if the directory is logically split. It can even be distributed to different server machines and locations.

In the following example (see Figure 51), the global directory of an organization has been split to form three directories located at different locations (circled in Figure 51). This may be done to address network performance and manageability issues. Also, two master servers are configured with replica servers.

Master servers A and B each have a replica server, E and D, respectively. These replica servers use a referral statement to point to their respective master servers for operations requested by clients that cannot be handled by the replica servers, such as updates to directory data. The referral statement is defined during the configuration of replica servers and is located in the slapd.conf file (see previous section 7.6, “Replication” on page 178). This kind of referral is called a *referral directive*, or Superior Reference (X.500). With LDAPV3, the directive in the configuration file is MasterServer for a replica server to point to a master server.

The referral directive is not only used to point to a master server for that particular replica server but also to point to servers that have a portion of the namespace or, if there is a hierarchy, to point to the uppermost server in the hierarchy.

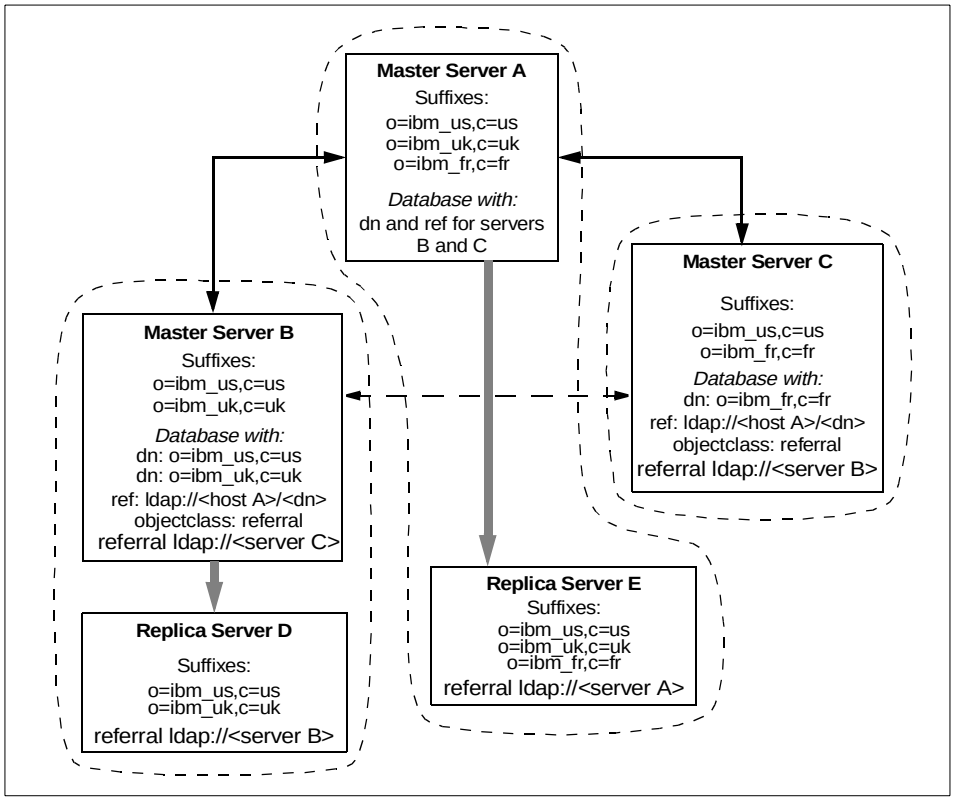


Figure 51. Referrals

Note that master server A (and its replica server E) does not actually contain any data; it only points to either server B or C (Figure 51).

To add such a referral directive, follow these steps:

1. Click on **Server** in the Navigation frame, then on **Properties**.
2. Type the hostname in the referral text field.
3. Click on the **Apply** button and restart the server to reread the configuration file.

The second element for LDAP referrals is called a *referral object* (subordinate reference in X.500). This type of referral is generally used to point downward in the DIT to other naming context. The requested DN is located in the server but does not have all the information to answer the request and returns referral information to the client instead. The referral returned will be with the

server reference and object as start point of the subtree containing information.

In an organization, for example, three suffixes could be defined on three servers. To form the complete DIT (Directory Information Tree), all three servers need to be linked. This task is accomplished with referrals. In this case, the referral is an entry of objectclass referral with ref as attribute. The ref attribute is the LDAP Web site of the referred entry on another LDAP server. The GUI or the command line utility ldif2db must be used to add such entries to the directory data.

Here is an example of an input files in LDIF format containing two referrals:

```
dn: o=ibm_fr,c=fr
ref: ldap://server_C/o=ibm_fr,c=fr
objectclass: referral
dn: o=ibm_uk,c=uk
ref: ldap://server_B/o=ibm_uk,c=uk
objectclass: referral
```

Do not forget to add the new suffixes to the respective server(s) before updating the database. Suffixes can be created by using the GUI under the Suffixes item in the navigation frame. On replica servers, the same suffixes must be added to get a correct replication (don't forget to restart the server after adding a suffix).

To verify whether referrals works in the scenario described above, the `ldapsearch` utility could be used as in the following example (refer to Figure 51):

On an LDAP client, this command could be run:

```
# ldapsearch -h <Server B> -b "o=ibm_fr,c=fr" cn=*
```

If the setup is correct, the utility will return all cn's under root `<o=ibm_fr,c=fr>`, which are not physically stored on server B but on server C because the referral points to server C. If the search concerns data on server B, one could type the command:

```
# ldapsearch -h <Server C> -b "o=ibm_us,c=us" cn=*
```

The utility would return all cn's under `<o=ibm_us,c=us>` physically stored on server B although the search was addressed to server C.

7.8 Command Line Utilities

Administration of the data in a directory is a sensitive point. Data administration concerns data management with a directory structure and content. The IBM SecureWay Directory server comes with a set of management utilities to assist an administrator in managing the directory's contents.

There are command line utilities that are used to handle large amounts of data, for example, to import thousands of entries in one single step. These utilities include the following:

- `ldif`
- `ldif2db`
- `bulkload`
- `db2ldif`

Because they access data in the database directly, the tools listed above must run on a directory server. Note that the `ldif2db`, `bulkload`, and the `db2ldif` utilities support the conversion from a specified local character set to/from UTF-8 (see also 4.3, "UTF-8 Support" on page 83).

Another set of utilities are more geared towards handling of single (or a few) entries at a time. These include:

- `ldapsearch`
- `ldapmodify`
- `ldapdelete`
- `ldapmodrdn`
- `ldapadd`

These latter tools use LDAP to communicate to an LDAP server, and they can, therefore, be run on a server or a client. They are, therefore, included in the IBM SecureWay Directory Client SDK. Source code for these utilities is provided, too, to allow application programmers to modify them as required or as sample programs to learn about LDAP programming. A parameter allows to specify whether LDAP Version 2 or Version 3 is to be used (the default is Version 2).

A brief explanation of each of the command line utilities follows. Additional information can also be found in the online documentation that is included with the product.

7.8.1 The LDIF Utility

LDIF (or LDAP Data Interchange Format) is a format for representing LDAP entries in text form (see also 4.7.1, “The LDIF File Format” on page 93). It is widely used and accepted as a de-fact standard, although it is, at the time of writing, only defined in an Internet Draft (see Appendix A, “Standards” on page 269). LDIF is used by the `ldapmodify`, `ldapadd`, and `ldapsearch` command line utilities, which are explained later in this chapter. It is also used as input to the `ldif2db` command, and it is the output format created by the `db2ldif` command. The `ldif` utility is a shell-accessible utility that converts arbitrary data values to a single attribute statement in LDIF. It reads input values from standard input and produces output in LDIF format.

The basic form for an entry in an LDIF file is as follow:

```
dn: <distinguished name>
objectClass: <object class>
objectClass: <object class>
<attrtype>: <attrvalue
<attrtype>: <attrvalue>
```

An LDIF file consists of a series of records separated by a blank line, and a record is a directory entry with a mandatory DN and at least an objectClass if the record is a new entry. If the record is for an update only, the DN is required. Some attributes, required by an objectClass, must be defined.

Attribute values can be clear text, such as a name, or it can be Base64 encoded binary data, such as for an JPEG picture. The `ldif` utility analyzes input data and senses the type of data for the correct output. The following example shows how `ldif` converts a JPEG file into one single attribute statement of an LIDF file:

```
# ldif -b jpegPhoto < Photo.jpg
jpegPhoto:: AQFjZCAvdXNyci9scGRhcAoAY2QgL3ci9sZGFwCgBjZCBldGMKAGxzCgBjZCBia
 W4KAGxzCgBjZCBjb25maWcKAABjZCAuLi9jb25WcKAGxzCgBjZCAuLi8KAGNkIHNiaW4KAABsc
 ...
 W51cy4qCgAAY2QgL3Vzci9sZGFwL2V0YwoAY2Q3Vzci9sZGFwLwoAAGxzCgBjZCBiaW4KAGxzC
 goAAGxkaWYgeHl6CgBscyAtbAoAAGxkaWYgLWIganBlZ1Bob3RDwuc2hfaGlzdG9yeQoA
```

The output created by `ldif` (shortened here in this example) is a Base64 encoded attribute line in LDIF that represents a JPEG photo. If the `-b` option is not specified and the input line contains only printable ASCII characters, then the generated output line(s) will not be Base64 encoded.

7.8.2 The LDIF2DB Utility

The `ldif2db` utility is used to load entries from a file in LDAP Directory Interchange Format (LDIF) into a directory. The database must already exist and so must the suffixes under which new entries are being added. The `ldif2db` utility may be used to add entries to an empty directory database or to a database that already contains entries.

The syntax is:

```
ldif2db -i <input file>
```

No additional parameters other than the filename of the LDIF input file are necessary since all other information (such as the suffix) is contained in the LDIF file.

7.8.3 The BULKLOAD Utility

Bulkload is used to load a large amount of data in LDIF format. The syntax is:

```
bulkload -i <input file> [-f <configuration file>]
```

The `-f` parameter may be used to specify an alternate slapd.conf configuration file. The default is <LDAP root directory>/etc/slpad.conf. The `bulkload` utility must not be executed while the server is running. A schema checking process verifies that individual directory entries are valid based on the object class definitions and attribute type definitions found in the configuration files(s). This will add some more time to load data, but it is the safest method to ensure that the data being loaded has the correct schema. Schema checking is controlled by the SCHEMACHECK environment variable. It can be set to YES, NO, or ONLY. The first two options control whether or not schema checking is performed during data import. The last, ONLY, specifies that no data shall be imported, but schema checking will be done with the data.

7.8.3.1 Tips to Consider Prior to Doing a Bulkload

The following tips are extracted from a White Paper that can be found at:

`http://www.ibm.com/software/network/directory/library/whitepapers/ldap_scaling.html:`

- On AIX, if the LDIF file you are loading contains a large number of entries, you may want to edit the /etc/security/limits and change the limits to unlimited (-1). Note that when you change them, you need to re-login for the change to take effect. The reason this must be done is that the bulkload tool consists primarily of scripts that parse the LDIF data into database ready files. In order to process large amounts of data, it is best to set the limits to unlimited. If you have set the limits to unlimited, and you

still have problems using `bulkload`, the only other alternative is to break up the LDIF file into smaller pieces.

- Make sure that you have the correct object class and attribute type definitions set up for the LDIF data that you are loading. These definitions can be viewed in the slapd.oc.conf and slapd.at.conf files. One approach that may be used to verify there are no problems with the data and definitions is to use the bulkload SCHEMACHECK=ONLY option first to validate the data and, thereafter, use the SCHEMACHECK=NO to load data into the directory.
- If you are adding a large amount of entries to your LDAP directory by doing multiple bulkloads, you may want to remove any indexes that you have defined in your database configuration. This will shorten the bulkload time. However, you must remember to redefine the indexes before using the LDAP directory. It is also important to note that when you change indexes in the database configuration, it will take the LDAP directory longer to start up. The reason for this additional time is due to dropping and recreating indexes. This will only happen the first time after indexes are changed.

You may consider backing up your existing database. This task can be done with the DB2 command `db2 backup db <dbname> to <file> or <tape>` (please see the DB2 documentation). Other DB2 command lines, such as `db2 connect to <your LDAP database>` or `db2 list database directory`, may be executed to verify if your database exists before loading any data.

7.8.3.2 Performing the Bulk Load

The following instructions assume that you have already created an LDAP directory and want to quickly add additional data entries using an LDIF file.

Set up the environment for bulkload

There are three environment variables that can customize the bulkload for your particular requirements:

```
# export SCHEMA_CHECK={YES|NO|ONLY}
```

If this variable is not specified, the default is YES, which means that schema checking will be done on the data prior to loading it into the directory. This will add additional time to the bulkload, but it is the safest method to ensure that data being loaded has the correct schema.

```
# export LDAPIMPORT=<temp dir>
```

This variable specifies a temporary directory that should have about 2.5 times the size of the LDIF file of free space. If this variable is not specified, the

default DB2 path will be used (/tmp/ldapimport on AIX). It is also important to note that the ldapdb2 user must have write permission to this directory.

```
# export DB2SORTTMP=/sortdir1,/sortdir2
```

This variable specifies one or more directories for temporary sorting files during the import. They should be large enough so that no errors occur during the import operation. If this variable is not specified, the default DB2 sort directory will be used (/u/ldapdb2/sqllib/tmp on AIX).

Perform the bulkload

Run the following command to perform the bulkload:

```
# bulkload -i <LDIF file> [-f config file]
```

7.8.3.3 The Bulkload Process

There are two major parts to the bulkload utility: Parsing the data, and loading the data.

Parsing the data

The first is the parsing phase, which parses through the LDIF file and prepares it to be loaded into the database. The path specified in the LDAPIMPORT environment variable is used to temporarily store the data to be loaded. If you would like to monitor progress on the parsing phase, you can look at the files being created in the LDAPIMPORT directory specified. There is one parsed file for every table that LDAP uses to store the directory information. The LDIF file is parsed linearly, and a separate file is created for each attribute. So, for example, if you had an LDIF file containing entries with a surname attribute (for example, surname: Miller), there would be a file created with the name surname in the LDAPIMPORT directory. Since the LDIF file is parsed linearly, you could perform a `tail -f` (AIX only) on the surname file in the LDAPIMPORT directory to find out the entry most recently parsed. This can give you an idea of how much has been processed and how much longer it may take to complete the parsing phase. If any errors occur during this phase of the bulkload, you can easily correct the error and restart the bulkload.

Loading the data

The second phase of the bulkload takes the parsed files and uses the db2 `load` command to load the data into the database. There is one parsed file for every table that LDAP uses to store the directory information. The ldap_entry table is the largest one, and is loaded first. This table will take about half of the total bulkload time; so, do not panic if this is taking a relatively long time. It is possible to monitor progress on the loading phase by typing the appropriate command for DB2 on your system to list the contents of

containers defined for User Data Space (USERSPACE). The user data space is used when you add entries to the directory. As long as there is recently current activity in the containers, the load is making progress. Since DB2 evenly spreads the data across the tablespace containers, you really only need to monitor one of the containers. Another good thing to do while the load is occurring is to monitor the available free space left on the containers. You do not want to run out of disk space while loading - as it is almost impossible to recover from!

Since it is almost impossible to recover from a failed partial load, it is a good idea to use the DB2 backup command fairly frequently when incrementally loading your directory.

7.8.4 The DB2LDIF Utility

This utility is used to dump entries from an LDAP directory into a text file in LDIF format. The syntax of the utility is:

```
db2ldif -o <output file> [-s subtree]
```

The `-o <output file>` option specifies the output file to contain the directory entries in LDIF. All entries from the specified subtree are written in LDIF to the output file. The file is created if it does not already exist; otherwise, it will be overwritten.

The `-s subtree` option identifies the top entry of the subtree that is to be dumped to the LDIF output file. If this option is not specified, all directory entries will be dumped to the output file based on the suffixes specified in the configuration file. The following example would extract all entries under `<o=ibm,c=us>` to a file named `output.ldif`:

```
# db2ldif -o output.ldif -s "o=IBM,c=us"
```

7.8.5 The LDAPSEARCH Utility

The `ldapsearch` utility is a command line utility built around the `ldap_search()` API. The utility opens a connection to an LDAP server, binds to the server, and performs a search using a specified search filter. If the request finds one or more entries, the requested attributes are retrieved, and the entries and values are printed to standard output. If no attributes are specified, all attributes associated with each returned entry are displayed.

The syntax is:

```
ldapsearch [options] filter [attributes]
```

The significant parameters for the `ldapsearch` tool are:

-b searchbase Use searchbase as the starting point for the search instead of the default. If -b is not specified, this utility will examine the LDAP_BASEDN environment variable for a searchbase definition. If neither is set, the default base is set to "".

-V Specifies the LDAP version to be used by `ldapsearch`. Specify -V 3 to run as an LDAP V3 application. This is implemented by using `ldap_init()` and `ldap_simple_bind_s()` API calls instead of the `ldap_open()` and `ldap_bind_s()`. Specify -V 2 to run as an LDAP V2 application, which is the default mode.

-n Shows what would be done, but it does not actually perform the search. This is useful for debugging in conjunction with the -v flag.

-v Runs in verbose mode with many diagnostics written to standard output.

-t Writes retrieved values to a set of temporary files. This is useful for dealing with non-ASCII values, such as jpegPhoto or audio data.

-A Retrieves attributes only (no values). This is useful when you just want to see if an attribute is present in an entry, and you are not actually interested in the specific values.

-B Does not suppress display of non-ASCII values. This is useful when dealing with values that appear in alternate characters sets, such as ISO-8859.1. This option is implied by -L (see below).

-L Displays search results in LDIF format. This option also turns on the -B option and causes the -F option to be ignored.

-R Specifies that referrals are not to be automatically followed.

-d debuglevel Sets the LDAP debug level to debuglevel.

-F sep Use sep as the field separator between attribute names and values. The default separator is '=', unless the -L flag has been specified, in which case, this option is ignored.

-f file Reads a series of lines from file, performing one LDAP search for each line. If file is a single '-' (dash) character, then the lines are read from standard input.

-D binddn Use binddn to bind to the LDAP directory. Use of binddn should be a string-represented DN, such as cn=root.

-w bindpasswd Uses bindpasswd as the password for simple authentication.

-h ldaphost Specifies an alternate host on which the LDAP server is running. The default is to use the local host on which the command is run.

-p ldapport Specifies an alternate TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified, and -Z is specified, the default LDAP SSL port 636 is used.

-Z Uses a secure SSL connection to communicate with the LDAP server.

-K keyfile Specifies the name of the SSL key database file (with a .kdb extension). If a key database filename is not specified, this utility will look for the presence of the SSL_KEYRING environment variable with an associated filename. Otherwise, no key database file will be used for server authentication, and default trusted certification authority roots will be used. The key database file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. This parameter is ignored if -Z is not specified.

-P keyfilepw Specifies the key database password. This password is required to access the encrypted information in the key database file. The encrypted information includes the set of trusted root certificates and, optionally, a private key with an associated client X.509 certificate. This parameter is ignored if -Z is not specified.

-N certificatename Specifies the label associated with the client certificate in the key database. Note that if the LDAP server is configured to perform server authentication only, a client certificate is not required. If the LDAP server is configured to perform client and server authentication, a client certificate is required. This parameter is not required if a default certificate/private key pair has been designated as the default. Similarly, certificatename is not required if there is a single certificate/private key pair in the designated key database. This parameter is ignored if -Z is not specified.

-s scope Specifies the scope of the search. The use of scope should be one of base, one, or sub to specify a base object, one-level, or subtree search. The default is sub.

-a deref Specifies how aliases dereferencing is done. The use of deref should be one of never, always, search, or find to specify that aliases are never dereferenced, always dereferenced, dereferenced when searching, or dereferenced only when locating the base object for the search. The default is to never dereference aliases.

-l timelimit Waits at most timelimit seconds for a search to complete.

-z sizelimit Limits the results of the search to most sizelimit entries. This makes it possible to place an upper bound on the number of entries that are returned for a search operation.

The search filter, in may cases, will either be a simple attribute search (such as cn=Smith) or for all attributes (cn=*). Search filters, however, can be fairly complex, and there is a separate RFC, RFC 2254, that you should refer to if you need all the details. The following is a brief description of search filters.

A 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 7 lists the operators for search filters.

Table 7. Search filter operators

Operator	Description	Example
=	Returns entries whose attribute is equal to the value.	cn=John Smith finds the entry with the 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 the following table, Table 8.

Table 8. Boolean operators

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

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.

Here is an example search:

```
# ldapsearch -b "o=ibm_fr,c=fr" cn="Robert Dean" cn telephonenumber
```

This will perform an anonymous subtree search on <o=ibm_fr,c=fr> for entries with a commonName (cn) of Robert Dean. The commonName (cn) and telephoneNumber values are retrieved and printed to standard output for any entries with commonName of Robert Dean as in this example:

```
cn=Robert Dean, ou=In Flight Systems, ou=LGE, o=ibm_fr, c=fr
cn=Robert Dean
telephonenumber=334-855-5703
```

Another example is to search a type of ACL (the following represents a single command line):

```
# ldapsearch -v -b "ou=Widget Division,ou=nhb,o=ibm_uk,c=uk" cn=* cn
ownerSource
```

This will perform a subtree search from the base specified by the -b parameter. All cns will be retrieved with their attributes values for cn and ownersource:

```
cn=Brenda England,ou=Widget Division,ou=nhb,o=ibm_uk,c=uk
cn=Brenda England
ownersource=OU=WIDGET DIVISION,OU=NHB,O=IBM_UK,C=UK

cn=David Delbert,ou=Widget Division,ou=nhb,o=ibm_uk,c=uk
cn=David Delbert
ownersource=OU=WIDGET DIVISION,OU=NHB,O=IBM_UK,C=UK
```

7.8.6 The LDAPMODIFY and LDAPADD Utilities

The `ldapmodify` utility is a command line utility built around the `ldap_modify()` API. The utility opens a connection to an LDAP server, binds to the server, and modifies an entry. The entry information is read from standard input or from a file. The `ldapadd` utility is implemented as a renamed version of `ldapmodify`. The `ldapadd` works the same way as the `ldapmodify` with the -a flag set.

The syntax of the utility is:

```
ldapmodify [options] [-f <ldif input file>]
```

The options are:

-a Adds new entries. The default for `ldapmodify` is to modify existing entries. If invoked as `ldapadd`, this flag is always set.

-b Assumes that any values that start with a '/' are binary values, and that the actual value is in a file whose path is specified in the place where values normally appear.

-c Continuous operation mode. Errors are reported, but `ldapmodify` will continue with modifications. The default is to exit after encountering an error.

-r Replaces existing values by default.

-n Shows what would be done, but does not actually modify entries. It is useful for debugging in conjunction with -v.

-v Uses verbose mode with many diagnostics written to standard output.

-F Forces application of all changes regardless of the contents of input lines that begin with *replica:* (by default, *replica:* lines are compared against the LDAP server host and port in use to decide if a replog record should actually be applied).

-R Specifies that referrals are not to be automatically followed.

-d debuglevel Sets the LDAP debug level to debuglevel.

-D binddn Uses binddn to bind to the LDAP directory. The use of binddn should be a string-represented DN, for example, cn=root.

-w passwd Uses passwd as the password for simple authentication.

-h ldaphost Specifies an alternate host on which the LDAP server is running. The default is to use the local host on which the command is run.

-p ldapport Specifies an alternate TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified, and -Z is specified, the default LDAP SSL port 636 is used.

-f file Reads the entry modification information from an LDIF file instead of from standard input. If an LDIF file is not specified, you must use standard input to specify the update records in LDIF format.

-Z Uses a secure SSL connection to communicate with the LDAP server. The -Z option is not supported by non-SSL versions of this tool.

-K keyfile Specifies the name of the SSL keyring file. If a keyring filename is not specified, this utility will look for the presence of the SSL_KEYRING environment variable with an associated filename. Otherwise, no keyring file will be used for server authentication, and default trusted certification authority roots will be used. The keyring file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. This parameter is ignored if -Z is not specified.

-P keyfilepw Specifies the keyring password. This password is required to access the encrypted information in the keyring file (including the private key). This parameter is ignored if -Z is not specified.

-N certificatename Specifies the label associated with the client certificate in the keyring file. Note that if the LDAP server is configured to perform server authentication, a client certificate is not required. If the LDAP server is configured to perform client and server authentication, a client certificate is required. This parameter is not required if a default certificate/private key pair has been designated as the default. Similarly, certificatename is not required if there is a single certificate/private key pair in the designated keyring file. This parameter is ignored if -Z is not specified.

Here is an example. Assuming you have to change some fields for a specific user, first get, with an `ldapsearch` command, all attributes associated with the cn you want to update:

```
# ldapsearch -b "o=ibm_us,c=us" cn="Robert Dean" >/tmp/change_dean
```

Then, change the content of the output file with new information:

```
cn=Robert Dean, ou=In Flight Systems, ou=Austin o=ibm_us, c=us
changetype: modify
replace: telephonenumber
telephonenumber: 334-9211-4444
```

or:

```
cn=Robert Dean, ou=In Flight Systems, ou=Austin o=ibm_us, c=us
changetype: modify
add: title
title: Grand chef
```

or:

```
cn=Robert Dean, ou=In Flight Systems, ou=Austin o=ibm_us, c=us
changetype: modify
delete: facsimiletelephonenumber
```

Finally, run the command:

```
# ldapmodify -D "cn=root" -w <passwd> -v -r -f /tmp/change_dean
```

This command, with a different content of the /tmp/change_dean file, will replace the value of the `telephonenumber` attribute, add a `title` attribute with a value of Grand Chef, or completely remove the `facsimiletelephonenumber` attribute from the directory entry.

An alternative syntax is supported for compatibility with older versions of `ldapmodify`. An attribute can be preceded by the '-' sign to remove it and the '+' sign to add the attribute with value. If you wanted to change an attribute value, you have to specify the line with a new value.

For example, the content of the LDIF input file is:

```
cn=Robert Dean, ou=In Flight Systems, ou=Austin, o=ibm_us, c=us
-facsimiletelephonenumber
```

This deletes the attribute `facsimiletelephonenumber`, and the following example changes the value for the `telephonenumber` attribute.

```
cn=Robert Dean, ou=In Flight Systems, ou=Austin o=ibm_us, c=us
telephonenumber= 334-9211-4444
```

Before modifying an entry, the option -n can be used with option -v to verify if the actions requested are those expected, no change will take place. You have to repeat the command without -n to apply the modification. The -D and -w options are to authenticate with the LDAP directory with a DN and password for single authentication. If -w is not given, the bind fails with the following error:

```
ldap_open( localhost, 389 )
ldap_bind: Invalid credentials
```

The success of a change can optionally be verified, for example, by using the administration GUI.

- Click on **Directory/Access control** in the Navigation frame and then on **Browse tree**.
- Click on the appropriate **+** sign to expand entries.
- Search the cn you want to verify and click on it to open a new window.
- Click on **Browse entry** in the Navigation frame of the new window to display all attributes associated with the cn.

Alternatively, you could verify the change with the `ldapsearch` utility.

7.8.7 The LDAPDELETE Utility

The `ldapdelete` utility is built around the `ldap_delete()` API. The utility opens a connection to an LDAP server, binds to the server, and deletes one or more entries. The distinguished names (DNs) of the entries to delete are read from standard input or from a file.

The syntax is:

```
ldapdelete [options] [-f <ldif input file>]
```

The options are:

-n Shows what would be done, but no entries are actually deleted. It is useful for debugging in conjunction with the -v option.

-b searchbase Uses searchbase as the starting point for the search instead of the default. If -b is omitted, the `ldapdelete` utility will examine the LDAP_BASEDN environment variable for a searchbase definition.

-v Uses verbose mode with many diagnostic messages written to standard output.

-c Continuous operation mode. Errors are reported, but `ldapdelete` will continue with deletions. The default is to exit after encountering an error.

-R Specifies that referrals are not to be automatically followed.

-d debuglevel Sets the LDAP debug level to debuglevel.

-f file Reads a series of lines from file, performing one LDAP delete for each line.

-D binddn Uses binddn to bind to the LDAP directory. The use of binddn should be a string-represented DN.

-w passwd Uses password as the password for simple authentication.

-h ldaphost Specifies an alternate host on which the LDAP server is running. The default is to use the local host on which the command is run.

-p ldapport Specifies an alternate TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified, and -Z is specified, the default LDAP SSL port 636 is used.

-Z Use a secure SSL connection to communicate with the LDAP server. The -Z option is not supported by non-SSL versions of this tool.

-K keyfile Specifies the name of the SSL keyring file. If a keyring filename is not specified, this utility will look for the presence of the SSL_KEYRING environment variable with an associated filename. Otherwise, no keyring file will be used for server authentication, and default trusted certification authority roots will be used. The keyring file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. This parameter is ignored if -Z is not specified.

-P keyfilepw Specifies the keyring password. This password is required to access the encrypted information in the keyring file (including the private key). This parameter is ignored if -Z is not specified.

-N certificatename Specifies the label associated with the client certificate in the keyring file. Note that if the LDAP server is configured to perform server authentication, a client certificate is not required. If the LDAP server is configured to perform client and server authentication, a client certificate is required. This parameter is not required if a default certificate/private key pair has been designated as the default. Similarly, certificatename is not required if there is a single certificate/private key pair in the designated keyring file. This parameter is ignored if -Z is not specified.

-dn Specifies one or more dn arguments. Each dn should be a string-represented DN.

Distinguished names (DNs) of entries to be deleted are read from standard input unless the -f <filename> was specified. It is recommended to first use the options -n and -v to be sure that the entry to be deleted is the right one as in the following example:

```
# ldapdelete -D "cn=root" -w pwd -n -v "cn=Harri Stranden, o=ibm_us, c=us"
!deleting entry cn=Harri Stranden, o=ibm_us, c=us

# ldapdelete -D "cn=root" -w pwd -v "cn=Harri Stranden, o=ibm_us, c=us"
ldap_open( localhost, 389 )
deleting entry cn=Harri Stranden, o=ibm_us, c=us
entry removed
```

This also works when the DNs to be deleted are read from a file (one DN to be deleted per each line).

```
# ldapdelete -D "cn=root" -w pwd -n -v -f /tmp/delete.cn
!deleting entry cn=Permana Widhiasta, ou=Austin, o=ibm_us, c=us
!deleting entry cn=Michel Melot, ou=Austin, o=ibm_us, c=us

# ldapdelete -D "cn=root" -w pwd -v -f /tmp/delete.cn
ldap_open( localhost, 389 )
deleting entry cn=Permana Widhiasta, ou=Austin, o=ibm_us, c=us
entry removed
deleting entry cn=Michel Melot, ou=Austin, o=ibm_us, c=us
entry removed
```

7.8.8 The LDAPMODRDN Utility

The `ldapmodrdn` utility is built around the `ldap_modrdn()` API. The utility opens a connection to an LDAP server, binds to the server, and modifies the relative distinguished name (RDN) of entries. The entry information is read from standard input or from a file.

The syntax is:

```
ldapmodrdn [options] [-f <ldif input file>]
```

The options are:

-r Removes old RDN values from the entry. Default is to keep old values.

-n Shows what would be done but does not actually change entries. This is useful for debugging in conjunction with the -v option.

-v Uses verbose mode with many diagnostic messages written to standard output.

-c Continuous operation mode. Errors are reported, but `ldapmodify` will continue with modifications. The default is to exit after encountering an error.

-R Specifies that referrals are not to be automatically followed.

-d debuglevel Sets the LDAP debugging level to debuglevel.

-D binddn Uses binddn to bind to the LDAP directory. The use of binddn should be a string-represented DN, such as cn=root.

-w passwd Uses password as the password for simple authentication.

-h ldaphost Specifies an alternate host on which the LDAP server is running. The default is to use the local host on which the command is run.

-p ldapport Specifies an alternate TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified, and -Z is specified, the default LDAP SSL port 636 is used.

-Z Uses a secure SSL connection to communicate with the LDAP server. The -Z option is not supported by non-SSL versions of this tool.

-K keyfile Specifies the name of the SSL keyring file. If a keyring filename is not specified, this utility will look for the presence of the SSL_KEYRING environment variable with an associated filename. Otherwise, no keyring file will be used for server authentication, and default trusted certification authority roots will be used. The keyring file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. This parameter is ignored if -Z is not specified.

-P keyfilepw Specifies the keyring password. This password is required to access the encrypted information in the keyring file (including the private key). This parameter is ignored if -Z is not specified.

-N certificatename Specifies the label associated with the client certificate in the keyring file. Note that if the LDAP server is configured to perform server authentication, a client certificate is not required. If the LDAP server is configured to perform client and server authentication, a client certificate is required. This parameter is not required if a default certificate/private key pair has been designated as the default. Similarly, certificatename is not required if there is a single certificate/private key pair in the designated keyring file. This parameter is ignored if -Z is not specified.

-f file Reads the entry modification information from file instead of from standard input or the command line (by specifying rdn and newrdn).

For example, assuming the file /tmp/rdn.dean contains the following:

```
cn=Robert Dean,ou=In Flight Systems,ou=Austin,o=IBM_US,c=US
cn=Mark Bold
```

The following `ldapmodrdn` command with the option -r to remove the old RDN would then change the RDN.

```
# ldapmodrdn -D "cn=root" -w its0 -r -v -f /tmp/rdn.dean>
ldap_open( localhost, 389 )
modrdn cn=Robert Dean,ou=In Flight Systems,ou=Austin,o=IBM_US,c=US:
cn=Mark Bold
removing old RDN
modrdn complete
```

7.9 Security Setup

One of the most important aspects of directory administration is determining the security policies. The security policy describes who is allowed to view and manipulate directory information.

Data within the directory will have varying levels of sensitivity. It is important that the more sensitive information be protected from unauthorized users. It is equally essential that administration not be limited to a single person unless absolutely necessary.

To facilitate installation and use, these issues should be addressed and resolved before introduction of a production system.

For LDAP projects, it is imperative that the security policy is understood and used by all applications. As an example, for objects and attributes that are shared (used) by more than one application, the security policy must carefully define who may create an object entry, modify, and delete the entry to avoid conflicts and broken pointers. To manage these rules, you have to use the LDAP Access Control List (ACL) concept.

The purpose of LDAP ACLs is to control information for a given object, and this control can be extended to all descendants of the object. If more than one users have control on this object, these users can be gathered in an *access group*, also some users can be members of a *role*. An access role is similar to an access group, but the access role has an implicit set of permissions on an object (see also 5.6, “Access Control” on page 120). This section presents how access groups and access roles are created and how to set ACLs for an object.

7.9.1 Creating and Working with Access Groups

Creating access control groups with an object class of AccessGroup is useful to assign a limited number of users the same authorizations. The main advantage of creating an access group is to minimize the ACLs administration because you do not have to manage each user’s individual access rights. Access groups can be managed through the DMT (Directory Management Tool) or using the administrator GUI.

Creating and working with access groups can most easily be done using the DMT. To add an access group, perform the following:

1. From within the DMT window, browse the directory (use **Tree** -> **Browse tree** from the menu tree) and expand the directory tree on the right as

necessary to select the entry DN to which you want to add an access group.

2. Click the **Add** button above the directory tree. A dialog opens that allows you to enter the RDN and an entry type for the new entry.
3. Type the RDN for the new access group (for example cn=Dept Acc Group) and click the **Access group** radio button. Then continue with **OK**.
4. On the upcoming dialog, enter a group name (for example Dept Acc Group) and the DN of one or more members. A description is optional.
5. Enter any other information as appropriate in the fields that appear after clicking on the **Other** tab.
6. Click **Create** to create the access group. The newly created access group will immediately be visible in the directory tree.

When using the administrator GUI rather than the DMT, follow these steps to create an access group:

1. Click on **Access groups** in the Navigation frame and then on **Create a new group**.
2. The upcoming panel offers you the choice of creating a completely new group or creating a group with the same members as an already existing group. Choose the option you need and click on **Next>**.
3. Select a suffix from the list proposed by the LDAP server.
4. Type a group DN in the Relative group DN*:* text field. An example could be 'cn=itso'. This DN will be automatically added to the suffix.
5. Type a group common name (CN) for the group in the Common name*:* text field. An example could be 'itso'.
6. Type a list of members that you initially want to be added to the access group in the Enter member names, one per line: text field.
7. Click on the **Finish** button after you have finished entering the information to create the group.

An access group will be created with a list of members as specified. Note that you cannot add a group without specifying at least one member.

To work with an access group, click on **Access groups** in the Navigation frame, and then select **Work with group**. Select the appropriate suffix and enter the group's RDN, then click **Work with access group**. A new browser window opens that lets you list, add, and delete members of that group. It also provides functions for deleting the group and to manage ACLs for that group (see 7.9.3, "Ownership and Access Control" on page 206).

By default, this new entry is created with object Class AccessGroup and has no ACL defined on it; the owner is inherited from the creator.

Once the access group has been created, you can return to the directory structure and assign this group to a specific entry. When a DN is removed from the directory, it is also removed from all access groups, access roles, ACL entries, and entry owners.

7.9.2 Creating and Working with Access Roles

Role-based authorization is a conceptual complement to the group-based authorization and its technical implementation, in fact, it is very similar. As a member of a role, you have the authority to do what is needed for the role in order to accomplish a job. Unlike a group, a role comes with an implicit set of permissions. There is not a built-in assumption about what permissions are gained (or lost) by being a member of a group.

Roles are similar to groups in that they are represented in the directory by an object. As such, roles contain a list of DNs. Roles that are to be used in access control must have an object class of AccessRole. The AccessRole objectclass is a subclass of the GroupOfNames object class.

Working with access roles is very similar to working with access groups (see above).

7.9.3 Ownership and Access Control

Access to LDAP directory objects and attributes is defined by Access Control Lists (ACLs). The set of attributes that are related to access control are manageable through the administrator GUI.

AclEntry A multi-valued attribute that describes access to attributes of the associated LDAP object as well as permissions on the object itself.

AclPropagate A true or false flag that indicates if the ACL should be propagated down the directory hierarchy.

AclSource A non-user modifiable attribute that identifies the directory object with which the ACL information is associated.

EntryOwner The owner of this particular directory object. The EntryOwner receives complete access to all attributes of the object. By making someone an entry owner, that person becomes an administrator for that entry. The entry owner is defined to be an ACL subject that has all rights

on the object and descendant if AclPropagate is set to yes.

OwnerPropagate A true or false flag that indicates if the object owner should be propagated down the directory hierarchy.

OwnerSource A non-user modifiable attribute that identifies the directory object with which the owner information is associated.

In general, an ACL is effective, and access is granted if the bindDN matches an access-id specified as an entryOwner or as administrator. If a bindDN matches an access-id specified within the aclEntry, the corresponding access rights apply. If the bindDN does not match any access-id, the granted rights correspond to the DN's group and role membership. If a match cannot be found, the requestor is granted anonymous rights.

7.9.3.1 Creating and Editing Ownerships

Ownerships and ACLs can be managed through command line utilities, the DMT, and through the administrator GUI. The following example uses the administrator GUI. To set the ownership of an object, proceed as follow:

1. Select the entry that is the root of the subtree you want to control with an ACL.
   - Click on **Directory/Access control** in the Navigation frame.
   - Click on **Browse tree** in the Working with entries in the directory work area.
   - Expand the tree as necessary and click on the object you want to manage.

   This opens a separate browser window where you can view entry attributes, manage access control, and search the entry's subtree.
2. In this new browser window, click on **Access control** and then on **Owners**. From the new work area Work with the access control owners of this entry, you can define the OwnerPropagate (by clicking the appropriate checkbox) and the OwnerSource attributes for this entry. If this is a new entry, you can only create the new entry's ownership by clicking on **Create entry ownership**. This brings up a panel as shown in Figure 52. Subsequent invocations of this function also lets you edit, delete, and remove ownership of an object.

   When adding additional owners after the initial ownership has been created, click on **Add,** and the new dialog lets you select from one of the following types of owners:
   - Local groups

- Local roles
- Local user
- Foreign user

Click the appropriate radio button, and the upcoming dialog requires you to select or enter the DN for the new owner. After finishing, click **Finish** to add the ownership. Proceed accordingly for editing, deleting, or removing object ownership. Note that you can have multiple owners for an object.

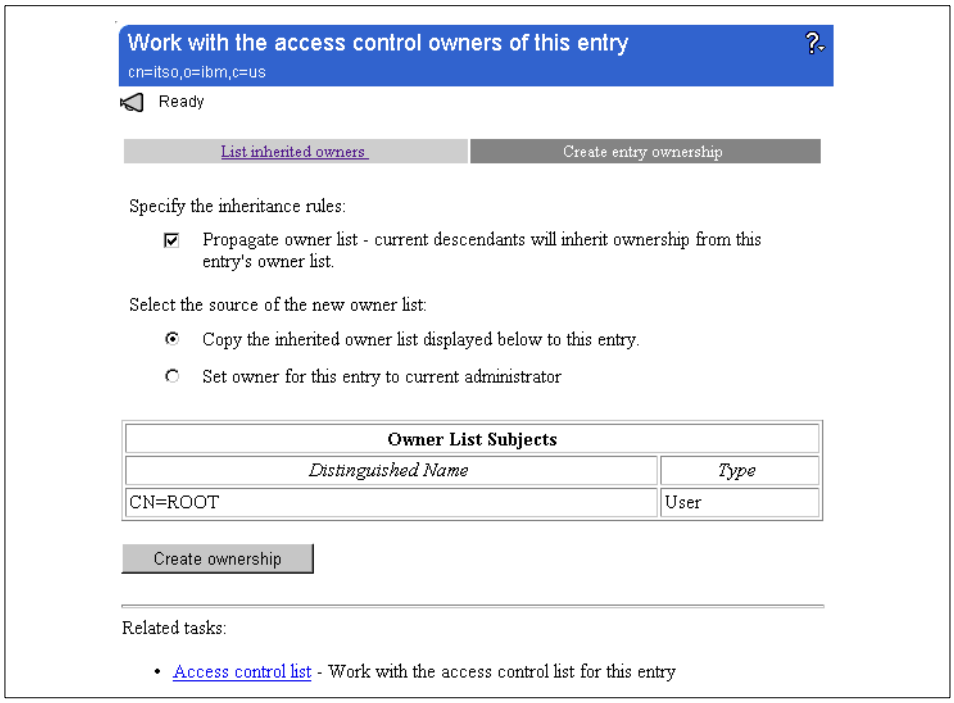


Figure 52. Create ownership

7.9.3.2 Managing ACLs

As with ownerships, ACLs can be managed using the command line utilities, the DMT, or the administrator GUI. The description below applies to the administrator GUI.

To define ACLs on an entry, click on **Access control list** in the Work with access control for this entry work area, and you will get a panel as shown in Figure 53.

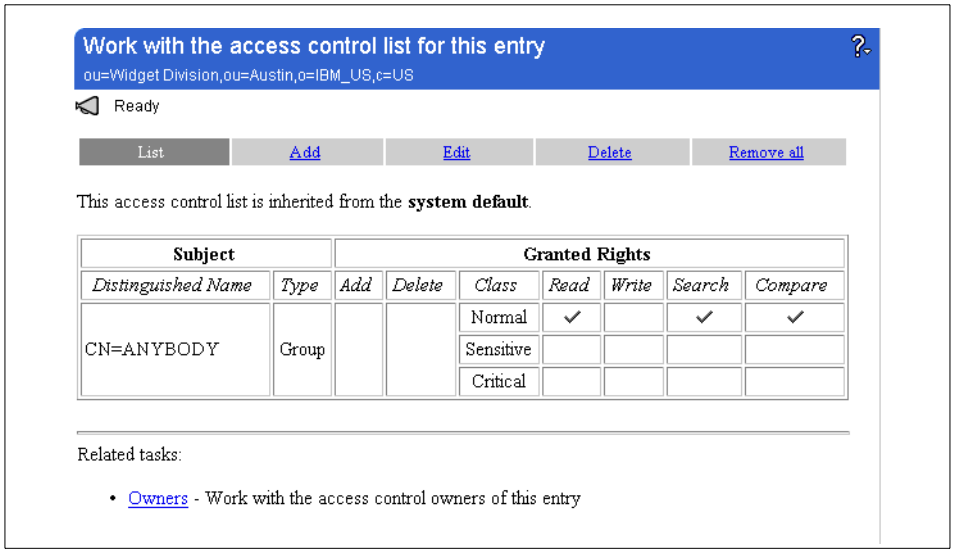


Figure 53. ACL control list entry

The default Access Control Group <cn=ANYBODY> (see Figure 53) is considered to be the group for all unauthenticated users. This group cannot be removed from the directory, and, by default, this group has read, search, and compare permissions to attributes within the normal class. If you remove <cn=anybody> from an ACL, all access will be suppressed for all users except for those defined under the modified entry.

Use the appropriate function (Add, Edit, Delete, or Remove all) from the functions list of the Work with the access control list for this entry work area (Figure 53). The subsequent dialogs guide you through the necessary steps. As with ownerships, an ACL object can be:

- local group
- local role
- local user
- foreign user

The instructions above used the administrator GUI to manage ACLs. As mentioned above, the DMT could be used for the same purpose, too (it might even be more intuitive to use). The following two figures, Figure 54 and Figure 55, show such an example using the DMT where a new ACL has been created and modified for <ou=ITSO,ou=Austin,o=ibm,c=us>. Figure 54 shows an additional group that has been added to the ACL with some more permissions than the Anybody group. Note that in the top portion of the

window it says *Source DN: OU=ITSO,OU=AUSTIN,O=IBM,C=US*. This is an indication that a specific ACL has been created for this DN; otherwise, it would only say *Source DN: default*.

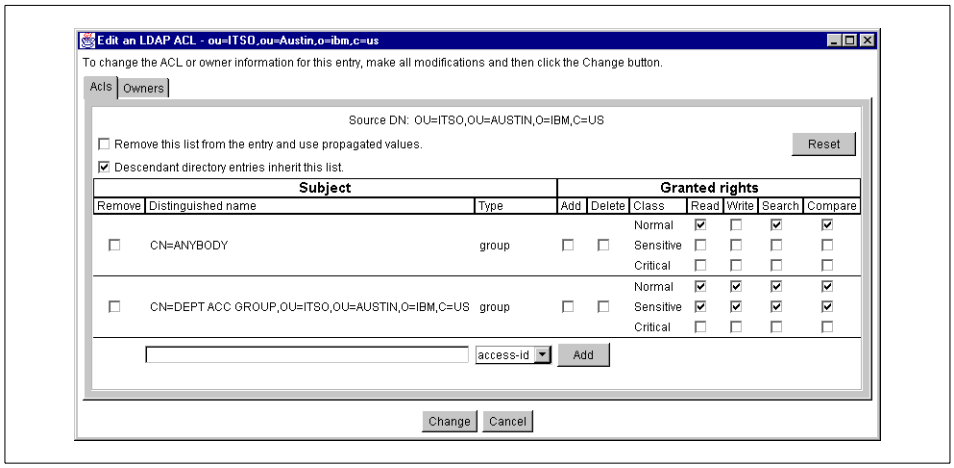


Figure 54. ACL with additional group

Figure 55 below shows an additional owner (John Smith) for the same ACL, which grants this DN all rights. The entry and ACL were created as cn=root, which is listed as well. Note that it is also possible to add a group or even a role as an ACL owner.

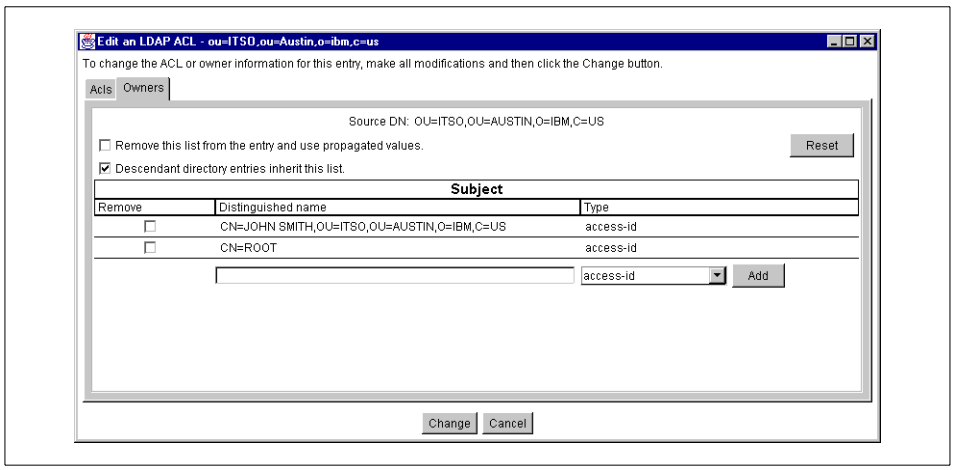


Figure 55. Additional ACL owner

The ACLs setting can be checked using, for example, the `ldapsearch` utility as shown in the following example:

```
# ldapsearch -b "ou=widget division,ou=austin,o=ibm_us,c=us" cn=* cn
aclentry entryowner
The utility will return a list of all cn's known under the specified base by
the command, with their aclEntry and enrtyowner attributes.
cn=Arthur Edwards, ou=Widget Division, ou=Austin, o=IBM_US, c=US
cn=Arthur Edwards
entryowner=access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK
entryowner=access-id:CN=ROOT,OU=IBM_US,C=US
aclentry=access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK:obj
ect:ad:normal:rwsc:sensitive:rwsc:critical:rwsc
aclentry=group:CN=ANYBODY:normal:rsc

cn=Curtis Edwards Jr, ou=Widget Division, ou=Austin, o=IBM_US, c=US
cn=Curtis Edwards Jr
entryowner=access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK
entryowner=access-id:CN=ROOT,OU=IBM_US,C=US
aclentry=access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK:obj
ect:ad:normal:rwsc:sensitive:rwsc:critical:rwsc
aclentry=group:CN=ANYBODY:normal:rsc
```

The LDIF output will also give you the following results for the modified entry:

```
dn: ou=Widget Division, ou=Austin, o=IBM_US, c=US
ou: Widget Division
objectclass: organizationalUnit
description: main product:Orange Widget Delux
businesscategory: home entertainment
ownerpropagate: TRUE
aclpropagate: TRUE
ownersource: OU=WIDGET DIVISION,OU=AUSTIN,O=IBM_US,C=US
aclsource: OU=WIDGET DIVISION,OU=AUSTIN,O=IBM_US,C=US
entryowner: access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK
entryowner: access-id:CN=ROOT,OU=IBM_US,C=US
aclentry: access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK:ob
ject:ad:normal:rwsc:sensitive:rwsc:critical:rwsc
aclentry: group:CN=ANYBODY:normal:rsc
```

And, for an entry that is a child of the previous entry (assuming ACL propagation was set on), the output is:

```
dn: cn=Mary Burnnet, ou=Widget Division, ou=Austin, o=IBM_US, c=US
objectclass: organizationalPerson
cn: Mary Burnnet
sn: Burnnet
telephonenumber: 1-812-855-5923
internationalisdnnumber: 755-5923
facsimiletelephonenumber: 1-812-855-5923
title: ISO Deputy, Qual. Tech
postalcode: 1515
seealso: cn=Linda Carlesberg, ou=Austin, o=IBM_US, c=US
aclsource: OU=WIDGET DIVISION,OU=AUSTIN,O=IBM_US,C=US
ownersource: OU=WIDGET DIVISION,OU=AUSTIN,O=IBM_US,C=US
aclpropagate: TRUE
ownerpropagate: TRUE
entryowner: access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK
entryowner: access-id:CN=ROOT,OU=IBM_US,C=US
aclentry: access-id:CN=DAVID CAMPBELL,OU=WIDGET
DIVISION,OU=NHB,O=IBM_UK,C=UK:ob
ject:ad:normal:rwsc:sensitive:rwsc:critical:rwsc
aclentry: group:CN=ANYBODY:normal:rsc
```

7.9.3.3 Attributes Access Classes

Attributes requiring similar permission for access are grouped together in classes. The three user modifiable attribute classes are:

- Normal
- Sensitive
- Critical

The administrator GUI lists them separately (see Figure 53) and provides check boxes for you to select ACL attribute values when editing ACLs for each attribute class separately.

If you prefer to use the command line utilities `ldapadd` or `ldapmodify` (with ACL information supplied in an LDIF file), keep the following considerations in mind:

- The aclSource and ownerSource attribute values are maintained by the server and cannot be modified by a user. If you try to set these values, you will receive an *insufficient access* error message.
- When running `db2ldif`, all ACL information for each node is included in the LDIF output. When this information is imported back into the database using `ldif2db`, the aclSource and ownerSource are used to determine if the ACL is inherited or explicit.

- You can use the `ldapmodify` command with either the add or replace options when setting an ACL (owner) if the ACL (owner) is inherited. If the ACL (owner) already exists, the replace option must be used.
- If an ACL (owner) is inherited, and the aclPropagate (ownerPropagate) is changed, the aclEntry (entryOwner) must also be specified.
- You cannot use the `ldapmodify` command's delete option for an inherited ACL (owner).
- The `ldapmodify` utility cannot be used to make changes, such as replacing aclPropagate and deleting the aclEntry, in the same operation. The ACL attributes on the object must remain in a legal state.
- When you use the `ldapmodify` add or modify option on an existing aclEntry, the new specification for the aclEntry becomes the new value. All previous values are deleted for that aclEntry object attribute. If your intent is to add additional information to an existing aclEntry, you must specify the entire existing entry in the operation.
- When you delete a user, that user is removed from all entries for aclEntry. accessGroups, and entryOwner positions. If you use the deleted DN to create another user, the new user will not have the same permissions as the previous user of the DN.

All ACLs attributes can be managed using LDIF notation. You can use `ldapmodify` with an LDIF file as input with these controls to change rights for a DN or to add a new subject.

- `changetype: modify` and `replace: aclentry`
- `add: aclentry`

The `ldif2db` utility can be used for similar tasks. You can find the syntax in 5.6, "Access Control" on page 120, as well as more information about ACLs.

7.10 Schema Data Management

A directory schema is a set of object class and attribute type definitions. The LDAP server uses this information to assure that the data stored in the directory is in accordance with these rules or to determine if and how an existing entry can be modified. The schema definitions are stored in files separate from the directory data that is stored in the underlaying relational database. Schema file management must be done carefully to get a reliable functionality of LDAP without corrupted data. This section explains the rootDSE and the files that are used by the IBM SecureWay Directory to store configuration and schema information.

7.10.1 The rootDSE

Although not directly related to the schema, it might be worth knowing that the IBM SecureWay Directory server (just as any other LDAP server) stores some information about itself. This is represented as a group of attributes located in the so-called *rootDSE*. These attributes are read-only, and they show up in the DMT as server properties, or they can be retrieved, for example, by performing a search of the root with a filter, such as objectclass=*. The server will return an entry with the following attributes:

- namingcontexts
- subschemasubentry
- altserver
- supportedextension
- supportedcontrol
- secureport
- supportedsaslmechanisms
- supportedldapversion
- referrals
- ibmdirectoryversion

The following is a description of the above attributes:

namingcontexts The values of this attribute correspond to naming contexts that this server masters or shadows. If the server does not master or shadow any information (for example, if it is an LDAP gateway to a public X.500 directory), this attribute will be absent. If the server believes it contains the entire directory, the attribute will have a single value, and that value will be the empty string (indicating the null DN of the root). This attribute will allow a client to choose suitable base objects for searching when it has contacted a server. It is the list of highest level suffixes defined for that server. For example, given the following two suffixes:

```
o=ibm_us,c=us
ou=Austin,o=ibm_us,c=us
```

Only the first one will be included in the list of the namingcontexts.

subschemasubentry The value of this attribute is the name of a subschema entry in which the server makes available attributes specifying the schema. The common setup is cn=schema.

altserver The values of this attribute are Web sites of other servers that may be contacted when this server becomes unavailable. If the server does not know of any other servers that could be used, this attribute will be absent.

supportedextension The values of this attribute are object identifiers (OIDs) identifying the supported extended operations that the server supports. If the server does not support any extensions, this attribute will be absent.

supportedcontrol The values of this attribute are object identifiers (OIDs) identifying controls that the server supports. If the server does not support any controls, this attribute will be absent.

secureport The value of this attribute is the port being used for secure connections, which is 636 for SSL.

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 will be absent. The IBM SecureWay Directory supports CDRAM-MD5 (see 5.2.1, “Overview of Simple Authentication and Security Layer (SASL)” on page 105).

supportedldapversion The values of this attribute are the versions of the LDAP protocol that the server implements. The supported LDAP versions of the IBM SecureWay Directory are 2 and 3.

referrals The values of this attribute are the URLs of other LDAP servers defined in referral objects in the local server.

ibmdirectoryversion The value represents the version of the IBM SecureWay Directory, which is currently 3.1.

Here is an example using the `ldapsearch` utility from the command line to query the rootDSE:

```
# ldapsearch -h ldap.itsc.austin.ibm.com -b "" -s base 'objectclass=*'
```

The server will return an entry as such:

```
namingcontexts=o=ibm_us,c=us
namingcontexts=o=ibm_nl,c=nl
namingcontexts=cn=localhost
subschemasubentry=cn=schema
altserver=ldap://ldap1.itsc.austin.ibm.com:389
supportedsaslmechanisms=kerberos_v4
supportedldapversion=2
supportedldapversion=3
referral=ldap://ldap2.itsc.austin.ibm.com
```

You can also get the same information by pointing your Web browser to an LDAP Web site:

```
ldap://<your_server>[:<port>]/
```

This will show these attributes in the browser's text area (provided your browser supports LDAP URLs).

7.10.2 Schema Files

In the previous release of the LDAP server, the IBM eNetwork LDAP Server that was based on LDAP Version 2, four files were present that stored the schema definitions. On AIX, for example, these are:

```
/etc/slapd.at.conf
/etc/slapd.at.system
/etc/slapd.oc.conf
/etc/slapd.oc.system
```

Two of these files are for object definitions (slapd.oc.*), the two others are for attribute definitions (slapd.at.*). The `slapd.oc.system` file contains object classes that the directory server requires, and it must not be modified. Therefore, administrators should not modify this file. The `slapd.at.system` file contains attributes that the directory server requires (for example, for operational and access control attributes), and it must not be modified. Administrators should not modify this file.

In Version 2.1 of the IBM eNetwork LDAP Server, administrators needed to edit these files in order to alter existing, or add new, schema definitions.

The IBM SecureWay Directory Server Version 3.1, supporting LDAP Version 3, uses more files that store schema definitions (refer to 2.5, "The IBM Schema" on page 41 for information about the standard schema). These files use a different syntax, and administrators should not edit them directly. These files are referenced to by the configuration directive `includeSchema` in the slapd.conf configuration file, such as:

```
includeSchema   /etc/V3.system.at
includeSchema   /etc/V3.ibm.at
includeSchema   /etc/V3.user.at
```

These three files, containing V3 system attributes type definitions, V3 IBM-specific attribute type definitions, and V3 user attribute type definitions, define attributes with more details than with V2. OID. Matching rules and syntax are added in these definitions. The IBM-specific attribute type definitions file (V3.ibm.at) may be superfluous in your directory context; the inclusion of this schema can be commented out by adding a hash ('#') at the beginning of the line.

The following three files contain V3 system object class definitions, V3 IBM-specific attribute object class definitions, and V3 user object class definitions:

```
includeSchema   /etc/V3.system.oc
includeSchema   /etc/V3.ibm.oc
includeSchema   /etc/V3.user.oc
```

As with the attribute type definition files, the object class definition files contain more information than was present with LDAP Version 2: The OID and a set of required (MUST) and optional (MAY) attribute types. If the IBM-specific object class definitions (V3.ibm.oc) are not used in your directory context, this schema can be disabled by adding a hash ('#') at the beginning of the line.

The following inclusion (and file) is to specify the V3 default syntax definitions:

```
includeSchema   /etc/V3.ldapsyntaxes
```

This inclusion (and file) specifies the V3 default matching rule definitions:

```
includeSchema   /etc/V3.matchingrules
```

The following `includeSchema` statement (and file) is for schema changes or new schema information. This statement must be the last `includeSchema` statement in slapd.conf file. The file is, by default, empty.

```
includeSchema   /etc/V3.modifiedschema
```

The slapd.conf configuration file contains another directive:

```
schemacheck V3_lenient
```

The `schemacheck` directive is used to specify the schema checking rules for add or modify operations. It can be set to *V2*, *V3*, *V3_lenient*, or *none*. *V2* keeps schema rules defined with Version 2; *V3* performs V3 checking. If *V3_lenient* is specified, not all parent object classes are needed, just the immediate object class is needed when adding entries. If *none* is specified, no schema checking will be performed, which is generally not recommended.

7.10.3 Back up and Restore Schema Information

Although the schema information can be queried using LDAP searches, it is not actually stored in the database but rather in separate files as explained above. For this reason, a backup and restore procedure for the directory server must include these files in addition to the directory data.

7.11 Locating LDAP Servers Using DNS

Different ways are available for an LDAP client to discover the existence and names (or IP addresses) of LDAP servers. Programming functions through the C or JNDI APIs can be used to specify server names (see, for example, Chapter 8, “Developing Directory-Enabled Applications” on page 223). This is useful when the clients have some way to store the server’s names (or IP addresses) locally, such as in a configuration file. This is an administrative burden if references to several LDAP servers are to be stored and maintained on a large number of clients. A simplification can be achieved when all clients only know one single LDAP master server. LDAP referrals on this master server can then be used to point requests to other LDAP servers. This master server must be configured to contain all suffixes available for the entire DIT and all referral objects with their ref attributes to point to the server with data requested.

Another approach is to use the Domain Name System (DNS), which gives you more flexibility as you do not need to configure each client, and it provides a central administration point to locate a suitable LDAP directory server.

DNS has become the standard lookup mechanism on the Internet and intranets for retrieving host addresses and other information. DNS servers need appropriate TXT, SRV, and CNAME records to handle LDAP servers. Also, DNS name servers need to use a version of BIND that supports the TXT and SRV record formats (greater than 4.8.3). API calls are provided for client application development that support DNS look-ups for LDAP servers. This API builds a list of LDAP servers with the first classed as the preferred or default server.

7.11.1 TXT Records

The TXT records associated with a server (or preceding address records) are used to store LDAP-specific server information. Four forms of TXT records are supported for an LDAP server entry:

service Provides an LDAP URL, which specifies host name, port, base DN, and security type (ldap for non-secure / ldaps for secure).

ldaptype Identifies replica server or master server.

ldapvendor Identifies server vendor (optional).

ldapinfo Provides any additional information, such as server contact, physical location, and so on (optional).

This is an example of a DNS record for an LDAP server with hostname `ldap` in the current domain:

```
ldap        A     11.22.33.44
            TXT   "service:ldap://ldap.us.ibm.com:389/o=ibm,c=us"
            TXT   "ldaptype:master"
            TXT   "ldapvendor:IBM"
            TXT   "ldapinfo: location Building 123 Austin"
```

7.11.2 SRV Records

SRV records allow an administrator to use several servers for a single domain to move services from host to host without disruption and to designate certain hosts as primary and other as alternate servers. The SRV records are optional except in cases where you have a server that supports SSL and non-SSL ports.

The SRV record has the following form:

```
service.proto.[name] [ttl] [class] SRV priority weight port target
```

`[ ]` denotes optional settings, the other fields are:

service Name of the desired service.

proto Protocol, tcp, or udp.

name Domain name associated with the record. If not specified, the default domain is assumed.

ttl Time-to-live, standard DNS meaning.

class Standard DNS meaning.

priority Target host with lowest number priority should be attempted first.

weight Load balancing mechanism when multiple hosts have the same priority. The chance contacting one of the hosts should be proportional to its weight and set to 0 if load balancing is not necessary.

port Port on the target port for the service. This value is ignored if the target host has a defined TXT record.

target Target host name.

For example:

```
ldap.itso.tcp     SRV   0 0 0 earth
                  SRV   0 0 0 earth_sec
                  SRV   1 1 0 venus
                  SRV   1 2 0 pluto
```

```
earth            A     11.22.33.1
                 TXT   "service:ldap://earth.itsc.ibm.com:389/o=ibm,c=us"
                 TXT   "ldaptype:master"
earth_sec        A     11.22.33.1
                TXT   "service:ldaps://earth.itsc.ibm.com:686/o= ibm,c=us"
                 TXT   "ldaptype:master"
venus            A     11.22.33.3
                 TXT   "service:ldap://venus.itsc.ibm.com:389/o=ibm,c=us"
                 TXT   "ldaptype:replica"
pluto            A     11.22.33.4
                 TXT   "service:ldap://pluto:389/o=ibm,c=us"
                 TXT   "ldaptype:replica"
```

In this example, a DNS search for ldap.itso with type=SRV would return four SRV records for three hosts. A SRV record is needed for each port/suffix combination supported by a server. In the example above, earth.itsc.ibm.com uses two ports, the standard port (389) and the SSL port (686). Thus, there are two SRV records and two TXT records for the same host and IP address.

7.11.3 CNAME Records

CNAME (canonical name) records allow you to define alias names for servers. Using a CNAME record, you could specify an alias, for example, LDAP in each domain that points to the real hostname of the LDAP server. The following example defines LDAP as being an alias for `server3.itsc.austin.ibm.com`:

```
LDAP        CNAME   server3.itsc.austin.ibm.com
```

LDAP clients in such a case could then just refer to the host LDAP in their own domain and get the correct address of whatever the LDAP server might be. Note that this is a standard DNS feature that does not need any special API calls on the client side.

7.11.4 APIs Provided for DNS Support

To make use of the DNS capabilities as described above, the following C language API calls are provided for client applications:

`ldap_server_locate()` Used to locate LDAP servers.

`ldap_server_free_list()` Used to free all storage associated with a linked list of server info structures.

`ldap_server_conf_save()` Used to store server information into the configuration file.

For JNDI (see 8.1, “Java Naming and Directory Interface (JNDI)” on page 223), the LDAPDNS class is available that provides the methods needed.

For more information on these API calls, we refer you to the online *LDAP Programming Reference* that comes with the product.

Chapter 8. Developing Directory-Enabled Applications

Directory exploitation is not exhausted after an organization's employees are stored in that directory where shrink-wrapped applications, such as Web browsers and e-mail clients, can look up e-mail addresses and telephone numbers. Directories can be used to store a much broader range of information, but it only makes sense as long as there are users for this information. The term *users* then not only applies to human individuals but also for applications and middleware software platforms that store and retrieve common information in a centralized directory. This is exactly the basic design idea behind LDAP; remember that LDAP defines a protocol that applications must use to communicate with a directory server. Applications use a common interface to that protocol. Two interfaces, also called Application Programming Interface (API), are provided with the IBM SecureWay Directory Client SDK for application developers: The Java Naming and Directory Interface (JNDI) and the C language API.

This chapter discusses the JNDI and C language APIs that enable clients, or more general applications, to access information stored in LDAP directories.

Please also refer to the online documentation that comes with the product, namely, the *JNDI Programming Guide* and the *C Programming Reference*. They can be found in the appropriate subdirectory of the IBM SecureWay Directory installation file tree (see Chapter 6, "Installation and Configuration" on page 131).

8.1 Java Naming and Directory Interface (JNDI)

JNDI, defined by Sun Microsystems, Inc., provides naming and directory functionality to Java programs. JNDI is an API independent of any specific directory service implementation. It enables seamless access to directory objects through multiple naming facilities.

The definition prevents, by design, the appearance of any implementation specific artifacts in the API. The unified API is designed to cover the common case. Providing this unified interface does not imply that access of unique features of a particular service, such as LDAP, is precluded, additional classes can be added to access service-unique features. JNDI can be used by a wide range of Java programs running on servers and traditional clients. JNDI can also accommodate a thin client by specifying a service provider that provides a proxy-style protocol where access to specific naming and directory services is relegated to a server. Security is dealt with by individual service providers; however, security related problems can be returned to the client.

8.1.1 Introduction

IBM provides an implementation of the JNDI where the service provider is LDAP backed by the IBM SecureWay Directory, which uses DB2 as the data store.

An application developer has two choices for accessing LDAP from a Java application. The Java LDAP API, sometimes called *JDAP*, is an LDAP class library defined in the Internet Draft *The Java LDAP Application Program Interface* (see Appendix A, “Standards” on page 269 for more details). For example, Netscape has implemented a Java API Software Development Kit (SDK) based on this draft. Sun Microsystems has developed the Java Naming and Directory Interface (JNDI) as part of its Java Enterprise API set, which also includes Enterprise Java Beans (EJB) and Java Database Connectivity (JDBC). JNDI is being supported by many vendors including IBM, Hewlett-Packard, and Novell.

Both, the Java API and the JNDI, support only a synchronous programming interface, though there can be multiple outstanding search enumerations occurring concurrently with other operations. However, a multithreaded Java application can continue processing while one thread is waiting on a synchronous LDAP call. The Java API closely follows the LDAP C API, while JNDI provides a generalized naming and directory interface. JNDI can access other directory services besides LDAP, such as the Network Information System (NIS), Novell Directory Services (NDS), and the Internet Domain Name System (DNS). Because of the wide acceptance of Sun’s Java Enterprise API, JNDI is commonly discussed as the Java interface to LDAP.

A naming service organizes and names objects. It provides an association known as a binding between a name and an object. The binding between a name and an object should not be confused with the connection between a client and a server, which is sometimes also called a binding. For example, a file system names and organizes files. The files are the objects that are bound to the names. Given a file name, the file itself can be retrieved. This model behind JNDI is independent of any name service for network components and services (such as LDAP, DNS, or Active Directory).

A directory service can be considered to be a specific type of naming service in which objects bound to names are directory entries. Directory entries are made up of attributes that store values describing the entity represented by the directory entry. The types of directory entries and attributes that can be stored are described by schema.

As discussed above, JNDI provides a generalized naming and directory service interface. For example, JNDI could be used to retrieve files from a file

system. In this case, a file system acting as a naming service could return the file that is bound to a particular file name. JNDI could also be used to access an LDAP directory, performing searches, and retrieving attributes.

JNDI represents an API that applications use to access a naming and directory service. The naming and directory service could be provided by any of a variety of such services, such as LDAP, NDS, or a file system. JNDI provides a Service Provider Interface (SPI) that enables access to the particular underlying directory service. The SPI is usually written by the vendor of the underlying naming and directory service and is supplied as a Java class library. This allows arbitrary services providers to be plugged into the JNDI Framework (see Figure 56).

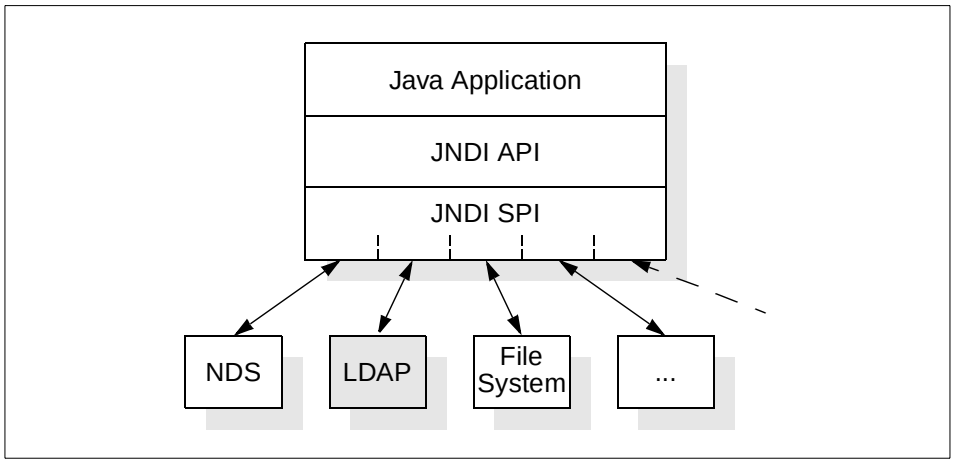


Figure 56. JNDI API and SPI interfaces

JNDI provides classes that implement a naming interface for applications, such as the file system example, that only look up names and access objects bound to names. JNDI also provides a directory interface that extends the naming interface. The directory interface adds functionality to access attributes and schema.

JNDI naming depends on syntactic rules or the naming convention of the underlying directory service. In JNDI terminology, a name is made up of individual components, called *atomic names*, that correspond to RDNs in LDAP. A sequence of atomic names is a compound name. JNDI naming depends on syntactic rules or the naming convention of the underlying directory service. An LDAP DN, for example, is a compound name. Since the underlying naming and directory services can have different name syntaxes, the SPI provides an implementation of a NameParser that can break a name

into its component parts. For example, LDAP RDNs are separated by commas; DNS names are separated by periods, and so on. Composite names are compound names that span different name spaces. For example, an LDAP URL can contain both a DNS and an LDAP name as, for instance, in `ldap://ldap.mycompany.com/cn=John%20Smith,o=ibm,c=us`.

8.1.2 Directory Context and Schema Context

Names are interpreted within a context. A context can be thought of as a particular node in the Directory Information Tree (DIT). If the current context is <o=ibm,c=us>, then the atomic name <ou=Austin> refers to the child node in the DIT with the DN <ou=Austin,o=ibm,c=us>. The node <ou=Austin,o=ibm,c=us> is also called a subcontext of <o=ibm,c=us>. A name space is traversed from context to subcontext, such as a file system is traversed from directory to the directory subtree as depicted in Figure 57.

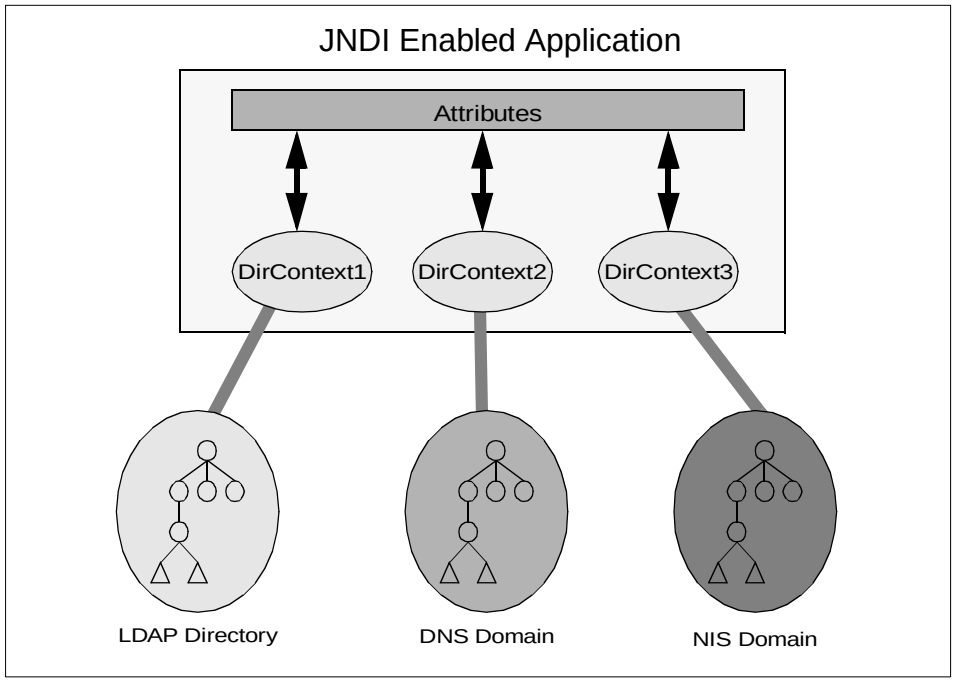


Figure 57. DirContext for different directory services

The DirContext interface extends the context interface by adding operations specific to a directory service, such as accessing attributes and searching. An application must establish an initial directory context as a starting point from which to do searches or traverse the DIT. The initial directory context is usually the name of an LDAP server.

LDAP V3 defines nine basic operations:

1. Bind
2. Unbind
3. Search
4. Delete
5. Modify
6. ModifyRDN
7. Add
8. Abandon
9. Extended

The IBM JNDI provides all of these operations of LDAP Version 3, except for abandon and extended operations, and controls either way through the DirContext Interface or through IBM specific extensions. Searches use a search filter as defined in *The String Representation of LDAP Search Filters*, RFC 2254. A SearchControls object passed to the search method can be set to control search characteristics such as the scope of the search, the number of entries returned, the time limit, and so on. Also, the entire schema name space can be browsed, and object and attribute schema definitions can be retrieved.

The IBM SecureWay Directory server can publish its schemas, and the IBM JNDI provides schema context to retrieve, view, and modify the server's schema. The context returned has the same hierarchical structure as a regular directory (schema tree) as shown in Figure 58 on page 228.

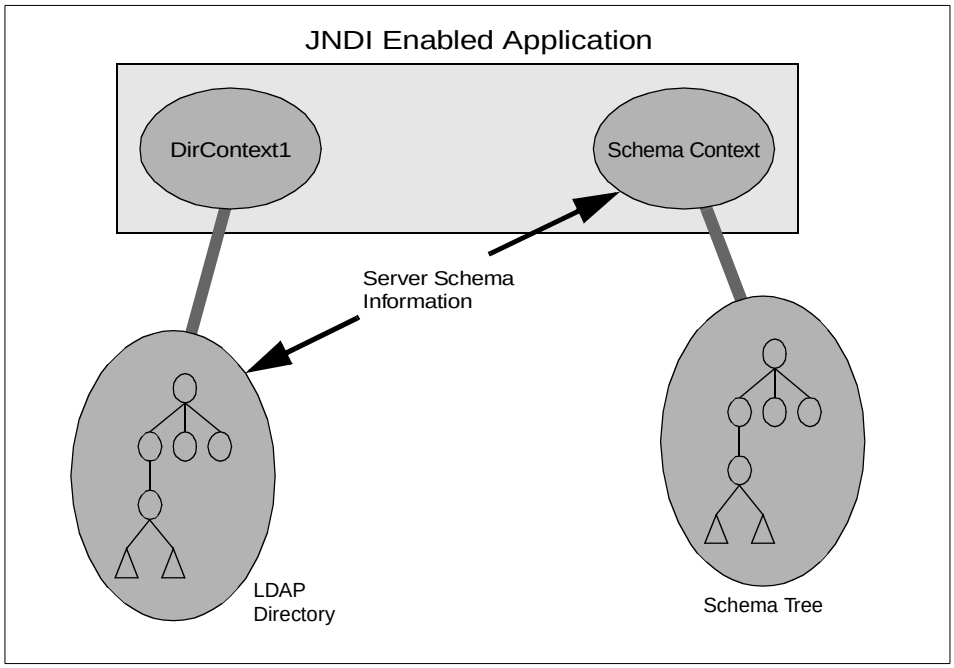


Figure 58. Schema context

When a directory context is established, it is passed through an environment that contains preferences and controls on how the directory service is accessed. The environment specifies the SPI to use, security level for binding to the server, and so on.

The environment is a hash table or properties list of key/value pairs. The environment settings could be coded in the application, retrieved from the system properties, or retrieved from a file. Table 9 lists some of the important environment properties. Different SPIs may support other environment properties and interpret or support values differently.

Table 9. JNDI directory context environment properties

Environment Property	**Use**
java.naming.factory.initial	Specifies the SPI.
java.naming.provider.url	LDAP URL that specifies the LDAP server.
java.naming.ldap.version	Specifies if server supports LDAP Version 2 or 3.

Environment Property	Use
java.naming.ldap.noBind	Specifies whether the client should bind to the server.
java.naming.referral	Specifies if referrals should be followed, ignored, or throw an exception.
java.naming.security.principal	Identity of user to authenticate.
java.naming.security.credentials	Password or other security credential.
java.naming.security.sasl	Class name of the SASL plug-in used to bind.

8.1.3 Java Object Serialization

The IBM JNDI classes support storage and retrieval of serialized java objects. This allows directories to be repositories for pre-initialized applets or beans, and this is possible outside the environment in which they are created, as shown in Figure 59 on page 229, which illustrates this feature.

Totally self-contained applets or beans can be downloaded from the directory. Advanced serialization features, such as externalizable interface and object input validation, are not supported.

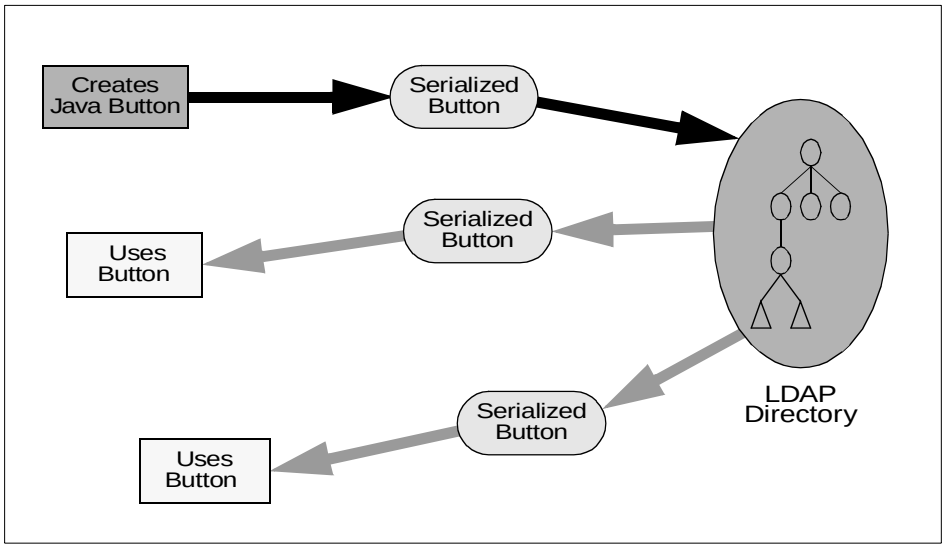


Figure 59. Java object serialization

8.1.4 JNDI and Security

JNDI does not specify a security interface or require any security model. The providers will define what security models they support. JNDI conveys any security information from an application to provider via a context's environment.

LDAP V3 allows a server to return data without binding. A clients can bind as an anonymous user with normally restricted search capabilities. The JNDI property that controls this behavior is `java.naming.ldap.noBind`:

```
env.put("java.naming.ldap.noBind","true")
```

No-bind operations are much faster, since no authentication handshake is performed between client and server, but not all the servers may support them. These operations are ideal for quick searches of non-sensitive information.

IBMs JNDI SPI supports two authentication models:

- Simple
- Simple Authentication and Security Layer (SASL)

When using a simple authentication, the password flows in clear text; the environment properties `java.naming.security.principal` and `java.naming.security.credentials` convey the bind DN and the password, respectively. If both properties are not specified, bind is anonymous:

```
env.put("java.naming.security.principal","cn=user,dc=ibm,dc=com");
env.put(Context.SECURITY_CREDENTIALS,"password");
```

IBMs JNDI also supports the authentication models named SASL (Simple Authentication and Security Layer), which is available in LDAP V3 mode. SASL is activated via plug-in classes, but not all servers may support SASL.

The `java.naming.security.sasl` property is used to identify the SASL mechanism to be used. SASL is a high-level security framework that allows different security mechanisms to be used and others to be plugged in (Figure 60, see also 5.2.1, "Overview of Simple Authentication and Security Layer (SASL)" on page 105).

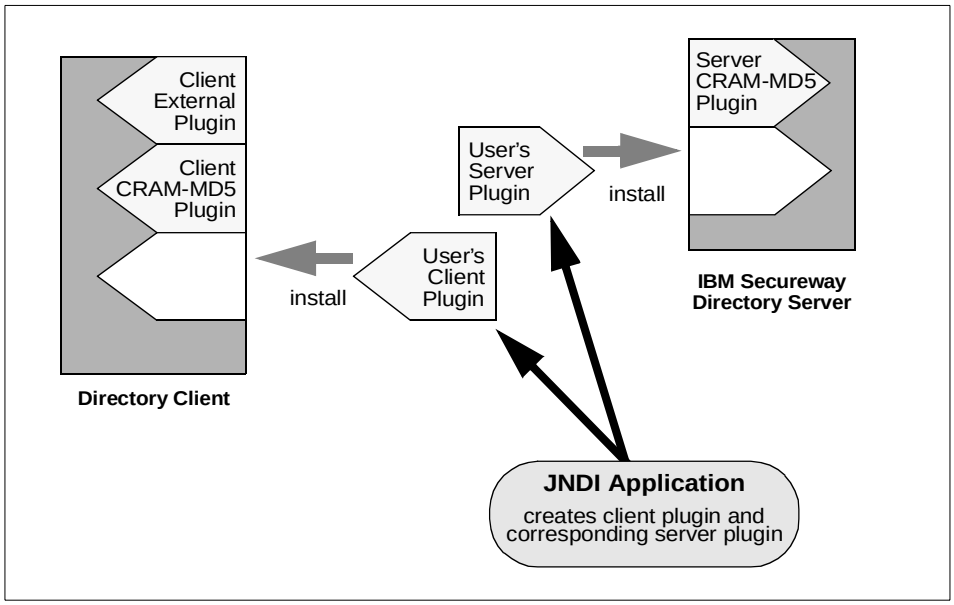


Figure 60. SASL plugins

IBMs JNDI supports two mechanism, which are:

1. The CRAM-MD5 SASL (com.ibm.ldap.LDAPSaslCRAM_MD5), which sends the DN and a hash of the password (shared secret) to the server for authentication using a challenge-response protocol.
2. SASL external (com.ibm.ldap.LDAPSaslExternal), which attempts to bind an underlying security protocol already negotiated, such as SSL (Secure Sockets Layer) client authentication. In most cases, the DN and password should be left uninitialized (Figure 61).

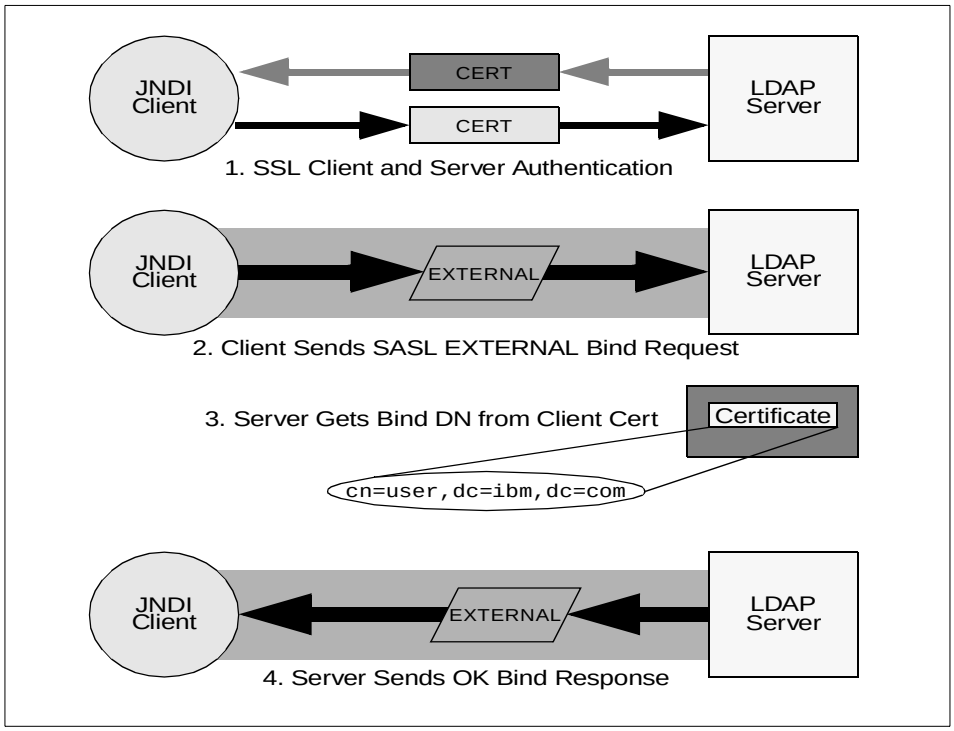


Figure 61. External SASL

IBMs JNDI SPI supports Secure Sockets Layer (SSL) for privacy. Currently, only SSLight is supported, other SSL packages are not supported at this time. It requires the IBM Global Security Kit (IBM GSKit) package for SSL (see also 5.3, “SSL Utilities” on page 108).

SSlight requires a keyring class file containing trusted signer certificates. This class may or may not contain a user private key to support client authentication (SASL External). The keyring file may, or should be, password protected if a private key is contained in it. The IBM GSKit contains a utility called ikmgui for managing and generating keyring files.

There are three special environment properties that have already been defined to configure the SSLight, which are:

1. The java.naming.security.ssl.keyring property (LDAPCtx.SECURITY_SSL_KEYRING), which is the fully qualified name of the above mentioned keyring class. If not specified, the default keyring class ldapkey will attempt to be loaded from classpath.

2. The java.naming.security.ssl.authentication property (LDAPCtx.SECURITY_SSL_AUTHENTICATION), which is the keyring password.
3. The java.naming.security.ssl.ciphers property (LDAPCtx.SECURITY_SSL_CIPHERS), which optionally lists the SSL ciphers. If not specified, the default set of ciphers are used.

8.1.5 JNDI Example Program

The following Java program uses JNDI to perform a search and print the attribute values of the directory entries found. It is a simple program that illustrates the basic ideas of:

- Setting up an environment and establishing an initial directory context.
- Setting up a search filter and search controls.
- Stepping through the returned entries and printing the values of the attributes.

The program is as such:

```
/*
 * Example JNDI program that performs an LDAP search
 * and parses and prints the results.
*/

import javax.naming.*;
import javax.naming.directory.*;
import java.util.Properties;
import java.util.Enumeration;

class Search  {

public static void main(String[] args) {

  try {
    /* Create an environment for the initial directory context.
       The properties specify the LDAP provider, the LDAP server,
       the LDAP version, and no security (anonymous bind). */

    Properties env = new Properties();
    env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
    env.put("java.naming.factory.url.pkgs", "com.ibm.jndi");
    env.put("java.naming.provider.url",
            "ldap://saturn.itso.austin.ibm.com");

    /* Create the initial directory context. */
```

```
    DirContext ctx = new InitialDirContext(env);

    /* Set up and perform the search.  Find all people in IBM in the
       United States whose common name starts with Sue or Johan. */
    String base = "o=IBM,c=US";
    String filter = "(|(cn=Sue*)(cn=Johan*))";
    SearchControls constraints = new SearchControls();
    constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
    NamingEnumeration results = ctx.search(base,filter,constraints);
    /* Print the search results. */
    if (!results.hasMore()) {
      System.out.println("Nothing found.");
    } else {
      /* For each entry found. */
      while (results.hasMore()) {
        SearchResult sr = (SearchResult) results.next();
        System.out.println(sr.getName());
        Attributes attrs = sr.getAttributes();
        if (attrs == null) {
          System.out.println("No attributes");
        } else {
          /* For each attribute of the entry. */
          for (NamingEnumeration ae = attrs.getAll(); ae.hasMore();) {
            Attribute attr = (Attribute) ae.next();
            String id = attr.getID();
            /* For each value of the attribute. */
            for (Enumeration vals = attr.getAll(); vals.hasMoreElements();
                System.out.println("   "+id + ": " + vals.nextElement()));
          }
        }
      }
    }
  } catch (NamingException e) {
    /* Handle any name/directory exceptions. */
    System.err.println("Search failed: " + e.getMessage());
  } catch (Exception e) {
    /* Handle any other types of exceptions. */
    System.err.println("Non-naming error: " + e.getMessage());
  }
}
}
```

The output of the program, executed against a sample directory is as follows:

```
cn=John Smith, ou=Austin, o=IBM, c=US
   sn: Smith
   title: ISO Deputy, Qual. Tech
```

```
   postalcode: 1515
   objectclass: organizationalPerson
   objectclass: person
   objectclass: top
   facsimiletelephonenumber: 1-812-855-5923
   telephonenumber: 1-512-838-6004
   internationalisdnnumber: 755-5923
   cn: John Smith
cn=Sue Kramer, ou=Austin, o=IBM, c=US
   sn: Kramer
   title: ISO Deputy, Qual. Tech
   postalcode: 1515
   objectclass: organizationalPerson
   objectclass: top
   facsimiletelephonenumber: 1-812-855-5923
   internationalisdnnumber: 755-5923
   telephonenumber: 1-812-855-5923
   cn: Sue Kramer
```

8.2 C LDAP Application Programming Interface (API)

This section describes the API library for C language applications. We use an example-driven approach to discuss the basic functions used for establishing connections, doing searches, and parsing the results. You will often find the LDAP server *saturn* mentioned in these examples. This is the name of the LDAP server that was used to run the examples running the IBM SecureWay Directory server on AIX.

8.2.1 Introduction

The basic conversation between an LDAP client and an LDAP server is essentially accomplished in four steps:

1. The first step is to initialize an LDAP session. This is done with the `ldap_init()` or `ldap_open()` API call, which returns a handle to an LDAP session allowing multiple sessions to be open at one time.
2. The next step is the authentication to the server. The `ldap_simple_bind_s()` and related functions are responsible for this. They support various authentication methods from simple authentication to the more sophisticated method Simple Authentication and Security Layer (SASL), available in LDAP Version 3.
3. Once the connection is successfully established, you can perform your LDAP operation(s), such as searching the directory for information and retrieving the results.

4. Finally, the connection has to be closed with the `ldap_unbind()` function call.

Here is the first, simple example that shows these basic steps:

```
/*
* example #1
*/

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

main()
{
        LDAP            *ld;
        char            *User = NULL;
        char            *Passwd = NULL;

/* initiate a connection */
if ((ld = ldap_init("saturn.itso.austin.ibm.com", LDAP_PORT)) == NULL) {
                fprintf(stderr, "ldap_init call failed !");
                exit(1);
}

/* authenticate as nobody (Passwd is NULL) */
if (ldap_simple_bind_s(ld, User, Passwd) != LDAP_SUCCESS) {
                ldap_perror(ld, "ldap_simple_bind_s");
                exit(1);
}

/*.............................
*   do something, for example
*   ask the server for information
*.............................*/

/* close and free connection resources */
ldap_unbind(ld);
exit(0);
}
```

The `ldap_init()` function takes as an argument the name of the LDAP server and the port where it is listening. The symbolic constant, `LDAP_PORT`, is set in the ldap.h file to 389. This is the default, non-secure port for LDAP. In case of success, `ldap_init()` returns a pointer to a data structure (a session handle), which contains information about the current session. It must be passed on to subsequent calls that refer to this session. In case of failure, `ldap_init()` returns NULL.

Note that ldap_init() does not actually open the session to the server, it only initializes it, and the session will only be opened when the first request is sent to the server. Therefore, you are able to change session settings as defined in the ID structure before the first connection occurs. When using ldap_init(), the first function that actually requires a connection will establish it automatically. In the example above, that is the ldap_simple_bind_s() function.

Instead of using ldap_init(), we could have used ldap_open() as well. It takes the same arguments, and it returns the same type of session handle. The difference between these two methods is that ldap_open() does open a connection to the LDAP server, while the ldap_init() only initializes a connection (see explanation above). The use of ldap_open(), however, is deprecated.

A look at the access log file of the LDAP server reveals that we have successfully connected (Netscape's Directory Server was used in this example).

```
...
[26/Mar/1998:14:22:42 -0600] conn=172 fd=35 slot=35 connection from 9.3.1.126
[26/Mar/1998:14:22:42 -0600] conn=172 op=0 BIND dn="" method=128 version=2
[26/Mar/1998:14:22:42 -0600] conn=172 op=0 RESULT err=0 tag=97 nentries=0
[26/Mar/1998:14:22:42 -0600] conn=172 op=1 UNBIND
[26/Mar/1998:14:22:42 -0600] conn=172 op=1 fd=35 closed
```

Notice that no user is listed in the dn field of the log file, which indicates that the client is authenticated as user *anonymous*. This is done by passing NULL values as user ID and password to the server within the ldap_simple_bind_s() instruction (see program example above). In case of success, ldap_simple_bind_s() returns LDAP_SUCCESS, an error code is returned otherwise. For more information about error handling, see 8.2.7, "Error Handling" on page 253.

Listed below are some of the session settings you can influence. You can find the complete list of possible options either in the API RFC or in the header file of the IBM SecureWay Directory Client SDK:

LDAP_OPT_SIZELIMIT	The maximum number of entries returned in a search; a value of LDAP_NO_LIMIT means no limit.
LDAP_OPT_TIMELIMIT	The maximum number of seconds spent on a search; a value of LDAP_NO_LIMIT means no limit.

LDAP_OPT_DEREF	The way to handle aliases. It can have one of the following values: LDAP_DEREF_NEVER, LDAP_DEREF_SEARCHING, LDAP_DEREF_FINDING, or LDAP_DEREF_ALWAYS. The LDAP_DEREF_SEARCHING means aliases should be dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_FINDING value means aliases should be dereferenced when locating the base object but not during the search.
LDAP_OPT_REFERRALS	Controls whether the LDAP library automatically follows referrals (LDAP_OPT_ON) or not (LDAP_OPT_OFF).
LDAP_OPT_HOST_NAME	The host name of the default LDAP server.
LDAP_OPT_ERROR_NUMBER	The number of the most recent LDAP error that occurred for this session.
LDAP_OPT_ERROR_STRING	The message returned with the most recent LDAP error that occurred for this session.

The way to set the session preferences depends on the SDK implementation you are using. In the IBM SecureWay Directory Client SDK, the connection handle is an opaque data structure that can only be accessed with the following two functions:

```
int ldap_get_option(
          LDAP            *ld,
          int             option,
          void            *outvalue);

int ldap_set_option(
          LDAP            *ld,
          int             option,
          void            *invalue);
```

Both functions return either LDAP_SUCCESS (integer value of zero) or a nonzero value, and the specified error code is set within the LDAP session handle. The option parameter specifies which session option is to be get or set. The invalue or outvalue parameters contain the new value for the option or the retrieved option value. This is a void pointer because the appropriate type depends on the option chosen. A short example follows for both of these functions.

To check whether or not the client automatically follows referrals returned from the LDAP server (default is yes), the code in the following example #2 could be used:

```
/*
* example #2
```

```
*/

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

main()
{
        LDAP            *ld;
        int             optdata;
        int             res;

/* initiate a connection */
if ((ld = ldap_init("saturn.itso.austin.ibm.com", LDAP_PORT)) == NULL)
                exit(1);

if (ldap_get_option(ld, LDAP_OPT_REFERRALS, &optdata) != LDAP_SUCCESS){
                ldap_perror(ld, "ldap_simple_bind_s");
                exit(1);
                }
else {
        switch(optdata){
                case LDAP_OPT_ON:
                        printf("Follow Referrals is activated\n");break;
                case LDAP_OPT_OFF:
                        printf("Don't Follow Referrals\n");break;
        }
}
exit(0);
}
```

The appropriate option to check is, as listed above, `LDAP_OPT_REFERRALS`. The result is captured in the integer variable, `optdata`, which is checked, subsequently, to find out which option is set.

To set the maximum number of seconds spend on each search, the `LDAP_OPT_TIMELIMIT` option has to be passed to the `ldap_set_option()` function:

```
/* Set number of seconds to spend on a search */
max_sec = 60;
if (ldap_set_option(ld, LDAP_OPT_TIMELIMIT,
                (void *)&max_sec) != LDAP_SUCCESS) {
        ldap_perror(ld, "ldap_set_option");
        exit(1);
}
```

8.2.2 Synchronous and Asynchronous Use of the API

You may have noticed the _s at the end of the bind command in example #1 above. This indicates that this command operates in synchronous mode with the LDAP server. We could have used `ldap_simple_bind()` (without a trailing _s) instead, then the communication would have been asynchronous. But what is the difference between the two modes?

The LDAP protocol allows you to handle multiple sessions at the same time. This means that several queries can be on their way to the server, and the order in which they are processed is up to the server. The LDAP protocol itself is, therefore, an asynchronous protocol.

In synchronous mode, the client sends a request to the server, and the function call only returns when it gets the reply from the server. It is blocked in between, which, in fact, means that no other operations can be processed. There is no message ID related to the request. The synchronous function returns either success or an appropriate error code.

When the client sends or receives requests in asynchronous mode, every message is tagged with a message ID that is unique for a given session. The client needs to use the function `ldap_result()` to check the status of the request and to get the results. The advantage of this approach is that the time gap between sending a request and actually getting the result from the server can be used by the client to do other work.

Figure 62 illustrates the differences between synchronous and asynchronous requests, and code samples are provided for each mode.

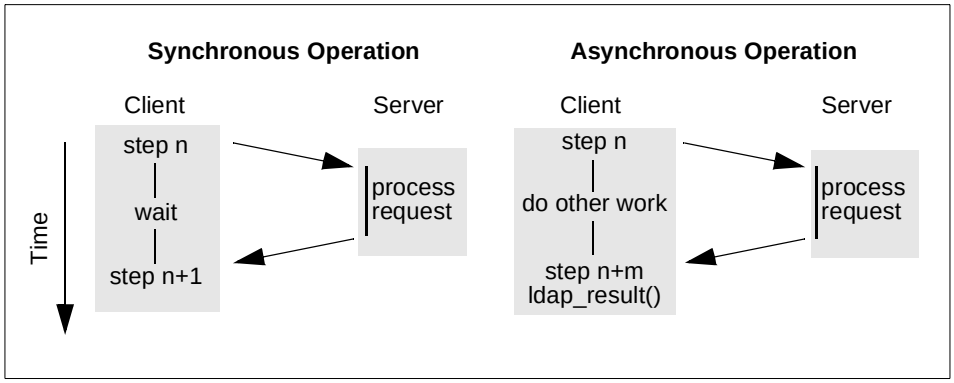


Figure 62. Synchronous versus asynchronous calls

Only functions that actually send data over the network are involved when comparing synchronous versus asynchronous functions. Each of these network-related functions, therefore, exist in either mode with an appended _s indicating the synchronous mode.

The synchronous approach is certainly not as powerful as the asynchronous mode. This means that complex operations with a large number of requests simply take more time. On the other hand, synchronous mode is much simpler to use. The method you chose, therefore, depends on what you intend to do.

8.2.3 A Synchronous Search Example

So far, the example for sample application above has only established a connection to an LDAP server but has not actually performed any operations. Now, we go a bit more into detail and provide a client that connects to the server and performs a search operation.

```
/*
* example #3
*/

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

#define SEARCHBASE      "o=IBM,c=US"

main()
{
        LDAP            *ld;
        LDAPMessage     *res;
        int             numfound;
        char            *User = NULL;
        char            *Passwd = NULL;
        char            line[BUFSIZ], search[] = "cn=";

/* Ask for the Name to search for */
printf("Type in a name to search for on LDAP server saturn:\n");
fgets(line, sizeof(line), stdin);
strcat(search, line);
search[strlen(search) - 1] = '\0';

/* initiate a connection */
if ((ld = ldap_init("saturn.itso.austin.ibm.com", LDAP_PORT)) == NULL)
        exit(1);

/* authenticate as nobody */
```

```
if (ldap_simple_bind_s(ld, User, Passwd) != LDAP_SUCCESS) {
        ldap_perror(ld, "ldap_simple_bind_s");
        exit(1);
}

/* search the database */
if (ldap_search_s(ld, SEARCHBASE, LDAP_SCOPE_SUBTREE, search, NULL,
    0, &res) != LDAP_SUCCESS) {
        ldap_perror(ld, "ldap_search_s");
        exit(1);
}

/* did we get anything ? */
if ((numfound = ldap_count_entries(ld, res)) == -1) {
        ldap_perror(ld, "ldap_count_entries");
        exit(1);
}

/* free memory allocated for search results */
ldap_msgfree(res);

/* close and free connection resources */
ldap_unbind(ld);

/* print the results */
printf("Found %d entries of name %s\n\n", numfound, line);
exit(0);

}
```

After asking for a name to search for from the command line, the program connects to saturn, the LDAP server, and searches for that name. The number of hits is then printed to the screen. Keep in mind that you are only able to find an entry if the access control of that entry is set appropriately. Otherwise, the entry may be there, but you are not allowed to search it.

There are three search functions: `ldap_search_s()`, `ldap_search_st()` and `ldap_search()` (two additional functions, similar to these three, are available for LDAP Version 3 that support server and client controls allowing an application to specify varying size and time limits for each search operation). The first two functions, `ldap_search_s()` and `ldap_search_st()`, are used in synchronous mode; the last one provides an asynchronous search function. In addition to the function of `ldap_search()`, `ldap_search_st()` lets you specify a time-out value for each search operation.

In the example above, `ldap_search_s()` was used. Its syntax is:

```
int ldap_search_s(
          LDAP      *ld,
          char      *base,
          int       scope,
          char      *filter,
          char      **attrs,
          int       attrsonly
);
```

The meaning of the parameters is as follows:

ld — The session handle obtained by ldap_init().

base — A DN that defines the starting point in the LDAP directory tree.

scope — This defines the way how a search in a tree is done. You can choose one of the three possibilities: LDAP_SCOPE_BASE, LDAP_SCOPE_ONELEVEL, LDAP_SCOPE_SUBTREE. See explanation that follows below.

filter — A character string as described in *The String Representation of LDAP Search Filters*, RFC 2254, representing the search filter.

attrs — A NULL-terminated array of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available attributes to be retrieved.

attrsonly — A Boolean value that should be zero if both attribute types and values are to be returned nonzero if only types are wanted. The latter option is useful if, you for example, you want to check to see if only a certain attribute is available.

As mentioned above, two additional search API calls were added in LDAP V3 that support server and client controls. These are ldap_search_ext() and ldap_search_ext_s(), respectively. Please read the online *Programming Reference* (supplied with the IBM SecureWay Directory Client SDK) for additional information.

To search an LDAP directory, a starting point in the tree structure of your directory hierarchy has to be defined. In our example, this is done by setting the base parameter for ldap_search_s() to "o=IBM,c=US".

Next, an appropriate scope needs to be chosen. There are three choices for selecting a search scope (see also Figure 63 on page 244):

- A particular entry (LDAP_SCOPE_BASE) can be searched for.
- The search can be extended to one level below the base not including the base (LDAP_SCOPE_ONELEVEL).

- The whole subtree under the starting point can be searched (LDAP_SCOPE_SUBTREE).

In the example above, we set scope to subtree (LDAP_SCOPE_SUBTREE) to look for entries containing the common name (cn) typed in by the user on the command line. The common name is appended to the attribute cn that is stored altogether in the variable search. This defines the simple search filter used in the example above. Because the attrs parameter is set to NULL and attrsonly is set to zero; all attributes with their values are retrieved.

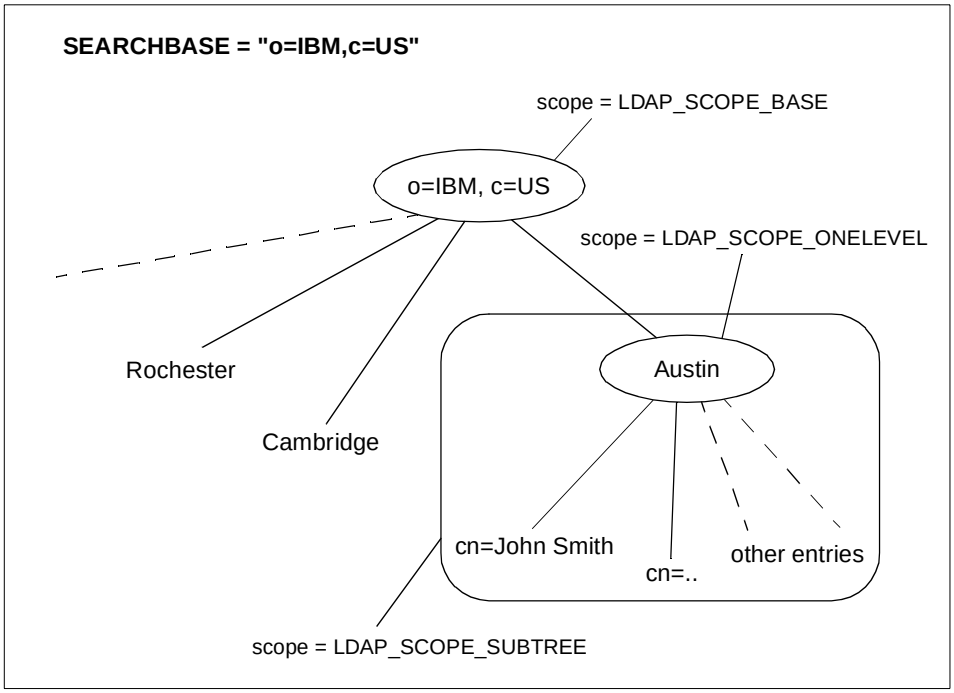


Figure 63. Different search scopes

The result of the directory search is returned by ldap_search_s() to a structure of type LDAPMessage pointed to by the res pointer. When no longer needed, the memory allocated in res should be freed using the ldap_msgfree() function (see sample code above). The function ldap_count_entries() is then used to count the number of entries found in the directory matching the search filter. Its syntax is:

```
int ldap_count_entries(LDAP *ld, LDAPMessage *result);
```

where:

ld is the connection handle, and

result is a pointer returned to the LDAPMessage structure filled by ldap_search_s() or ldap_result()

In case of an error, ldap_count_entries() returns -1, otherwise, the number of entries found.

The function ldap_msgfree(LDAPMessage *result) should be used to free the memory occupied by the search results. When successful, the result type freed is returned. This would be LDAP_RES_SEARCH_ENTRY in our case.

8.2.4 More about Search Filters

We used the filter "cn=common name" to look up directory entries. But the filter parameter of the search functions is much more flexible. Its syntax is defined in *The String Representation of LDAP Search Filters,* RFC 2254.

The basic syntax of the search filter is as follows:

```
(attribute operator value)
```

So, when we use the filter cn=John Smith, cn is the attribute, the equal sign is the operator, and John Smith is the value. There are several more operators available, for example, comparison operators, such as smaller than (<=) or greater than (>=). Furthermore, you can combine several filters using Boolean operators and, thus, search, for example, for more than one attribute.

For a complete explanation of search filters, please study the above mentioned RFC. Search filters are also introduced in more detail under 7.8.5, "The LDAPSEARCH Utility" on page 192.

8.2.5 Parsing Search Results

The outcome of a search request is usually a chain of entries as shown in Figure 64 on page 246. The last example program only counted the number of entries. But that is usually not what we want; we are interested in the information itself. Therefore, we need to parse the information returned by the server. This is an iterative process that starts at the very outside of the data container (an entry) and digs itself deeper in the data structure until it eventually gets to the single attribute/value pairs. The functions we need are:

```
LDAPMessage *ldap_first_entry(LDAP *ld, LDAPMessage *result);
LDAPMessage *ldap_next_entry(LDAP *ld, LDAPMessage *preventry);
```

where:

ld is the connection handle;

result is a pointer to the data structure obtained by ldap_search_s(), ldap_search_st(), or ldap_result();

preventry is a pointer to an entry returned by a previous ldap_first_entry() or ldap_next_entry(), and

return value if successful, returns a pointer to the first (ldap_first_entry()) or to the next entry (ldap_next_entry()), or NULL in case of no more entries or an error.

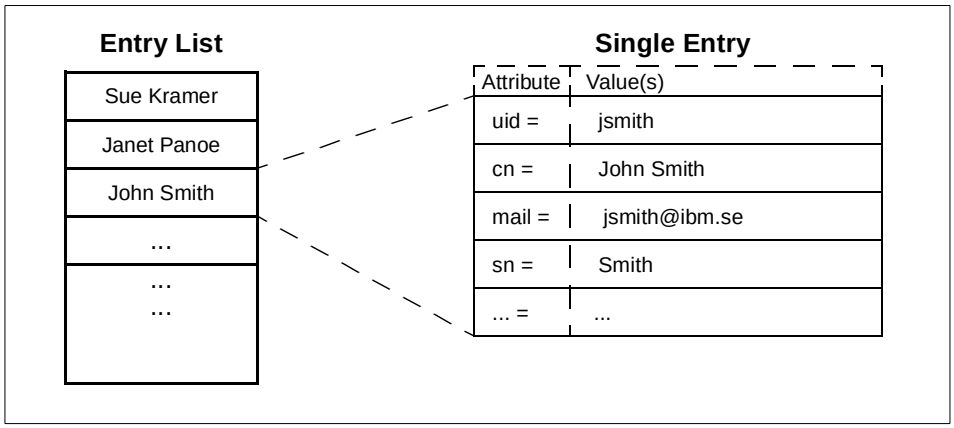


Figure 64. Result of a search request

But to actually retrieve the contents of an entry, we need to go further on. As outlined in Chapter 2, “Schema and Namespace” on page 31, an entry of an LDAP directory consists of attribute/value pairs as defined in the object classes. The following functions retrieve the name of single attributes:

```
char *ldap_first_attribute(LDAP *ld, LDAPMessage *entry, BerElement **ber);
char *ldap_next_attribute(LDAP *ld, LDAPMessage *entry, BerElement *ber);
```

where:

ld is the connection handle;

entry is a pointer to an structure returned by ldap_first_entry() or ldap_next_entry(), and

ber is a pointer to a data structure that is used to keep track of the current attribute. BerElement refers to data encoded using the Basic Encoding Rules. This pointer needs to be passed to subsequent calls of ldap_next_attribute().

Now that we have the attribute names, we are ready for the last step. We can retrieve the attribute values. The function used depends on the attribute types. If they consist of string data, the function:

```
char **ldap_get_values(LDAP *ld, LDAPMessage *entry, const char *attr);
```

can be used. If the attribute contains binary data, such as images in JPEG format, we have to use:

```
struct berval **ldap_get_values_len(
                      LDAP *ld, LDAPMessage *entry, const char *attr);
```

The parameters ID and entry are the same as in `ldap_first_attribute()` and `ldap_next_attribute()`. The parameter `attr` is a character string returned by `ldap_first_attribute()` and `ldap_next_attribute()`. Example #4 below shows a function called `check_result_and_print()` using the functions described above to parse search results:

```
/*
* example #4
*/

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

void check_result_and_print(LDAP *ld, LDAPMessage *res){

LDAPMessage     *entry;
BerElement      *ptr;
int             numfound, i;
char            *dn, *attr, **vals;

/* did we get anything? */
if ((numfound = ldap_count_entries(ld, res)) == -1) {
    ldap_perror(ld, "ldap_count_entries");
    exit(1);
}

/* parse the results */
if (numfound > 0) {

    /* for each entry print out dn plus attributes */
    for (entry = ldap_first_entry(ld,res); entry != NULL;
                            entry = ldap_next_entry(ld,entry)) {

            /* check for distinguished name */
            if((dn = ldap_get_dn(ld,entry)) != NULL){
                    printf("\n\ndn: %s\n", dn);
```

```
                ldap_memfree(dn);
            }

            /* get the attributes */
            for (attr = ldap_first_attribute(ld, entry, &ptr);
                        attr != NULL;
                        attr = ldap_next_attribute(ld, entry, ptr)) {
                printf("%s: ", attr);

                /* print each value */
                vals = ldap_get_values(ld, entry, attr);
                for (i = 0; vals[i] != NULL; i++) {
                printf("%s, ", vals[i]);
                }
                /* print the end of line for each attr. */
                printf("\n");
                ldap_value_free(vals);
            }
    printf("\n");
    }
} else {
    /* print that we didn't get anything */
    printf("Nothing found!\n");
}

/* free the search results */
ldap_msgfree(res);
}
```

This function takes as input the connection handle `ld` and a pointer to a structure `res` as returned by the `ldap_search_s()` function. First, it checks using the `ldap_count_entries()` function, what `ldap_search_s()` has returned. Next, if the number of entries is greater than zero, it starts parsing the results.

The attribute/value pairs are printed out in LDIF format (as described in 4.7.1, “The LDIF File Format” on page 93). To be LDIF compliant, the first line of every entry has to be the distinguished name (DN). The `ldap_get_dn()` function looks it up for us. Its syntax is:

```
char *ldap_get_dn(LDAP *ld, LDAPMessage *entry);
```

The parameters `ld` and `entry` are the same as in the `ldap_next_entry()` function. After retrieving the DN, we use the function `ldap_memfree()` to free the memory occupied by `ldap_get_dn()`. You may also notice the function `ldap_value_free()`. This is used to free memory blocked by the array that contains the attribute values. Its return value is void.

8.2.6 An Asynchronous Search Example

So far, we have only dealt with synchronous functions. We now change our search example in order to make it communicate asynchronously with the LDAP server. As mentioned in 8.2.2, “Synchronous and Asynchronous Use of the API” on page 240, the function `ldap_search()` is used for that purpose. It only initiates the search and, therefore, does not directly return the results. Instead, it returns a message ID, which identifies the search being processed by the server and serves as a parameter for the `ldap_result()` function, which can, subsequently, be used to check the search results.

```
int ldap_search(LDAP *ld, const char *base,
                int scope, const char* filter,
                char **attrs, int attrsonly);
```

Note that the pointer to the result structure is missing as a parameter here as compared to the example shown in 8.2.3, “A Synchronous Search Example” on page 241. The other parameters behave like the ones already mentioned as we described `ldap_search_s()`. Here is the example code:

```
/*
* example #5
*/

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

#define SEARCHBASE      "o=ibm,c=US"

/* prototype */
void check_result_and_print(LDAP *ld, LDAPMessage *res);

main()
{
        LDAP            *ld;
        LDAPMessage     *res;
        char            *User = NULL;
        char            *Passwd = NULL;
        char            line[BUFSIZ], *filter, temp[BUFSIZ];
        int             msgid, rc, i;
        struct timeval  tv = {0, 0};
strcpy(temp, "cn=");

/* ask for the Name to search for */
printf("\nType in a name to search for on LDAP server saturn:\n");
fgets(line, 20, stdin);
```

```
strcat(temp, line);
temp[strlen(temp) - 1] = '\0';
filter = temp;

/* initiate a connection */
if ((ld = ldap_init("saturn.itso.austin.ibm.com", LDAP_PORT)) == NULL)
                exit(1);

/* authenticate as nobody */
if (ldap_simple_bind_s(ld, User, Passwd) != LDAP_SUCCESS) {
                ldap_perror(ld, "ldap_simple_bind_s");
                exit(1);
}

/* search asynchronously */
if ((msgid = ldap_search(ld, SEARCHBASE, LDAP_SCOPE_SUBTREE,
                filter, NULL, 0)) == -1){
        ldap_perror(ld, "ldap_search_s");
        exit(1);
}

/* Initialize the value returned by ldap_result() */
rc = 0;
i = 0;
while (rc == 0) {

        /* ... do other work */
                printf("Loopcount %i\n",i++);
                        /* while waiting ... */
        /* check the status of the search operation */
        rc = ldap_result(ld, msgid, 1, &tv, &res);

        switch(rc) {
            case 0:
                /* do nothing, search is still in progress */
                break;
            case -1:
                /* some error occurred */
                ldap_perror(ld, "ldap_result");
                exit(1);
                break;

            case LDAP_RES_SEARCH_RESULT:
                /* result is complete, print it */
                check_result_and_print(ld, res);
                break;
        }
```

```
}

/* close and free connection resources */
ldap_unbind(ld);

exit(0);

}
```

Checking of the search results is done within the while() loop using the ldap_result() function. This function can be generally used to retrieve results of asynchronous search functions. The return value can have one of three general values. If it is -1, then some sort of error has occurred. A value of zero indicates a time-out, and a value greater than zero indicates a successful returning of a result. The various possible positive return values are declared in the ldap.h file. Only three of these return values are relevant to the search function. The reminder of possible return values belongs to several other asynchronous functions used, for example to add, modify, compare, or delete LDAP tree values. The (positive) return values associated with ldap_search() are:

LDAP_RES_SEARCH_ENTRY
: A single entry matching a previously initiated search result.

LDAP_RES_SEARCH_RESULT
: Either a result indicating the final outcome of a previously initiated search operation or an entire chain of entries matching the search operation along with the final outcome.

LDAP_RES_SEARCH_REFERENCE
: When the search result is a reference (this was added in LDAP Version 3).

For reasons of completeness, here are the other result values. Apart from the last one, the names should be self-explanatory:

- LDAP_RES_BIND
- LDAP_RES_MODIFY
- LDAP_RES_ADD
- LDAP_RES_DELETE
- LDAP_RES_MODDN
- LDAP_RES_COMPARE
- LDAP_RES_EXTENDED (new in LDAP Version 3, this is the return of a protocol extensibility operation)

The syntax of `ldap_result()` is:

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

where:

ld — is the connection handle.

msgid — This is the return value of a previously issued asynchronous function, in our case, ldap_search(). If you specify the constant LDAP_RES_ANY (-1), then the result of any operation is requested.

all — A Boolean parameter that is only used in search operations. If it is set to zero (false), only one message at a time is retrieved; if it is set to non-zero (true), all results should be received before returning them in a search chain.

timeout — A structure that specifies how long to wait for results to be returned. It takes the parameter tv_sec and tv_usec, which specifys seconds and micro seconds of the time interval. A NULL value causes ldap_result() not to return until results are available. A zero value (numeric 0, not to be confused with NULL) specifies a polling behavior. This means if values are available, ldap_result() retrieves them immediately; if not, it will not wait.

res — A pointer to the result obtained by asynchronous operations. This memory area should be freed with ldap_msgfree() when it is no longer needed.

Through the setting of the parameter *all* to *true* in example #5, we specified that we want all results retrieved at once. The time-out period of zero seconds (see the tv structure in the sample code) causes a polling behavior. Therefore, every call of `ldap_result()` checks whether the search operation has already finished. If not, it returns (with value zero in that case) and other work can be done within the while loop. In our case, it simply prints out the number of loops already processed. If `ldap_result()` returns `LDAP_RES_SEARCH_RESULT`, this indicates that the final outcome of the search operation, and that the chain of results is available. The result processing is then done by the formerly introduced function (see example #4) `check_result_and_print()`.

As mentioned at the beginning of this section, this is the big advantage of the asynchronous method. Once the asynchronous command is transferred to the server, the client is free to do other things. It uses the `msgid` and `ldap_result()` to check the outcome of the operation whenever appropriate.

We specified in the above example #5 by setting the parameter *all* to *true* that all results should get returned all together. This may be inconvenient, especially when a large number of entries may be expected. In setting the

parameter *all* to *false*, we can cause `ldap_result()` to deliver single search entries and not the whole result chain. This frees the client from waiting until the complete result chain is available. Every time a new result entry is available, `ldap_return()` delivers `LDAP_RES_SEARCH_ENTRY` asa return value instead of `LDAP_RES_SEARCH_RESULT`. However, the latter value indicates the end of the result list and delivers the final outcome. The following lines change the result processing behavior as just described; the rest of the code is the same as in example #5:

```
/* Initialize the values */
rc = 0;
i = 0;

/* while the search is still in progress, do this */
while (rc != LDAP_RES_SEARCH_RESULT) {

        /* ... do other work; print loop count in this example */
                printf("Loopcount %i\n",i++);
                        /* while waiting ... */
        /* check the status of the search operation */
        rc = ldap_result(ld, msgid, 0, &tv, &res);

        switch(rc) {
            case -1:
                /* some error occurred */
                ldap_perror(ld, "ldap_result");
                exit(1);
                break;
            case LDAP_RES_SEARCH_ENTRY:
                check_result_and_print(ld, res);
                break;
            case 0:
                /* no result yet */
                break;
        }
}
```

8.2.7 Error Handling

If an LDAP function fails, information about what went wrong can be found in the connection handle. Most of the error codes, which go into three separate fields of the connection handle, are directly returned from the server, but the fields can get set from client library functions as well. The fields are:

ld_matchedIn the event of an LDAP_NO_SUCH_OBJECT error, this parameter contains the part of the DN that could be matched with a DN found on the server.

ld_error This parameter contains the error message sent in the result by the server.

ld_errno This is the LDAP error code, such as LDAP_SUCCESS, LDAP_NO_SUCH_OBJECT, LDAP_STRONG_AUTH_REQUIRED, and so forth, indicating the outcome of the operation.

How the error processing works depends on what LDAP function you use. Most functions (all synchronous functions) directly return numerical error codes. They can get mapped by the function `ldap_err2string()` to character strings, which, in turn, can get printed to standard error output or to whatever is appropriate for the application.

When, for example, searching the directory server using the synchronous function `ldap_search_s()`, error checking can be done like this:

```
/* search the database */
if ((rc = ldap_search_s(ld, SEARCHBASE, LDAP_SCOPE_SUBTREE, search, NULL,
                        0, &res)) != LDAP_SUCCESS){
        fprintf(stderr, "ldap_search_s: %s\n", ldap_err2string(rc));
        exit(1);
}
```

In case of success, `ldap_search_s()` returns `LDAP_SUCCESS` (which is equivalent to zero) or the appropriate numerical error code. If we set the `SEARCHBASE` parameter to a nonexistent value, for example <o=icm,c=us> instead of <o=ibm,c=us>, then the error message of the previous example would be as follows (the particular return code equals to 32, `LDAP_NO_SUCH_OBJECT`):

```
ldap_search_s: No such object
```

This indicates that the entry we were looking for does not exist either due to an incorrect DN or for other reasons.

A common way to monitor the return code of an LDAP function for errors is also to use the function ldap_perror(LDAP *ld, char *msg). In fact, as you might have noticed, that is what we did in our examples so far. This function was defined in the LDAP Version 2 API. It internally converts the error contained in the ld_errno field of the session handle to an error string and prints it together with the msg string to standard error (stderr). In LDAP Version 3, the use of this function is deprecated. Therefore, using the ldap_err2string() function should be the preferred way in LDAP Version 3.

When checking for an error of an asynchronous function, ldap_parse_result() has to be used. This is because the function that checks the outcome of the

operation, ldap_result(), returns the type of the result (for example LDAP_RES_ADD, LDAP_RES_SEARCH_ENTRY, and so on) instead of the LDAP error code.

The routine ldap_parse_result() checks messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE returned from the LDAP server when looking for a result message to parse. It returns the constant LDAP_SUCCESS if the result was successfully parsed and no error was found; otherwise, it returns another error code. This function then also sets the appropriate fields in the connection handle. The syntax of `ldap_parse_result()` is as follows:

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

The parameters are:

ld, res — The connection file handle and the pointer to the message structure which contains the result of an LDAP operation as returned by ldap_result().

errcodep — An integer pointer that will be filled with the error code of the LDAP operation. That is the way the server tells the client about the outcome of its operation. NULL may be passed to ignore this field.

matcheddnp — In case of an LDAP_NO_SUCH_OBJECT error, this parameter will be filled with the part of the distinguished name that could be matched. NULL may be passed to ignore this field. The memory area occupied by this parameter should be freed using the ldap_memfree() function.

errmsgp — This result parameter will be filled in with the contents of the error message contained in the returned message. NULL may be passed to ignore this field. The memory area occupied by this parameter should be freed using the ldap_memfree() function.

referralsp — This parameter will be filled in with the contents of the referrals field contained in the returned message indicating zero or more alternate LDAP servers where the information should be retrieved. The referrals array should be freed by calling ldap_value_free(). NULL may be passed to ignore this field.

serverctrlsp This result parameter will be filled in with an allocated array of controls copied out of the LDAPMessage structure. The occupied memory area should be freed using ldap_controls_free().

freeit This determines whether or not the LDAPMessage structure is cleared after extracting the necessary information. Pass a non-zero value to free it.

Note that ldap_parse_result() places the error code in the errcodep parameter. Thus, check this parameter to trace errors of previous LDAP operations.

If we apply this to our last example, the asynchronous search example, we could do the following to check for errors after the `ldap_search()` has been invoked:

```
/* while the search is still in progress, do this */
while (rc != LDAP_RES_SEARCH_RESULT) {

        /* ... do other work; print loop count in this example */
                printf("Loopcount %i\n",i++);
                        /* while waiting ... */
        /* check the status of the search operation */
        rc = ldap_result(ld, msgid, 0, &tv, &res);
        switch(rc) {
            case -1:
                /* some error occurred */
               fprintf(stderr, "ldap_search_s: %s\n", ldap_err2string(rc));
                exit(1);
                break;
            case LDAP_RES_SEARCH_ENTRY:
                check_result_and_print(ld, res);
                break;

            /* this is the end of the search, test for errors */
            case LDAP_RES_SEARCH_RESULT:
                ldap_parse_result(ld,res,&err,&errdn,&errmsg,NULL,NULL,1);
                if (err != LDAP_SUCCESS) {
                        fprintf(stderr,
                                "Search Error: %s, %s, Matched DN:%s\n",
                                errmsg, ldap_err2string(err), errdn);
                        ldap_memfree(errdn);
                        ldap_memfree(errmsg);
                }
                break;
            case 0:
```

```
                /* no result yet */
                break;
        }
}
```

As we pointed out earlier, when the all parameter in ldap_result() is set to zero (false), single search results get retrieved one at a time. The final outcome is stored in the message structure when LDAP_RES_SEARCH_RESULT is returned. The function mentioned above, ldap_parse_result(), is then used to check the final outcome of the search. The error code is stored in the integer value err. A return value not equal to zero (LDAP_SUCCESS) indicates a search error. In this case, ldap_err2string() is used to transform the error number to the related error string. This is, together with errmsg and matcheddn, then printed out to the standard error output. The ldap_memfree() eventually frees no longer needed memory areas.

If we searched for an entry with the same wrong search base as in an earlier example above (o=icm,c=us), we would get the error message:

```
Search Error: , No such object, Matched DN:c=us
```

This corresponds to an `LDAP_NO_SUCH_OBJECT` error. The errmsg field is not set, but the matched DN field shows us the portion of the name that could successfully be matched.

Another function for error checking is available in the LDAP Version 3 API. It is ldap_parse_sasl_bind_result(). It allows an application to check for errors resulting from a SASL bind operation. For more information about this function, we refer you to the online *Programming Reference* that comes with the product.

8.2.8 Authentication Methods

Authentication can be understood as identifying a client (or a user) to the server. This needs to be done before any operation can be performed with the server. There are several authentication mechanism supported in LDAP. We start with Basic Authentication that was introduced in 5.2, “Security Support of the IBM SecureWay Directory” on page 104.

LDAP Version 3 includes the SASL authentication framework. A common way to gain a higher level of security, especially when exchanging information over the Internet, is to use SSL to encrypt the session, which is a method supported by SASL. The use of SASL/SSL will be explained in this section as well.

For general information about the different security techniques and the definitions and terms used below, we refer you to Chapter 5, “Directory Security” on page 103.

If you not only want to read and search a directory but also want to modify the entries in it, the anonymous user authentication used so far to search the directory will certainly not be sufficient. You need to properly authenticate a user who needs special access permissions, for example as an authorized directory administrator. The easiest way to do this is to use basic authentication. This is done by specifying a DN and a password in the `ldap_simple_bind()` function and sending it over to the LDAP server as in the following example:

```
/*
* example #6
*/

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

main()
{
        LDAP            *ld;
        char            *User = "cn=Directory Manager";
        char            *Passwd = "1234qwer";

/* initiate a connection */
if ((ld = ldap_init("saturn.itso.austin.ibm.com", LDAP_PORT)) == NULL)
                exit(1);

/* authenticate to the server */
if (ldap_simple_bind_s(ld, User, Passwd) != LDAP_SUCCESS) {
                ldap_perror(ld, "ldap_simple_bind_s");
                exit(1);
}

printf("Authenticated as %s\n", User);

/* ..............................
*   do something, for example
*   ask the server for information
* .............................. */

/* close and free connection resources */
ldap_unbind(ld);
exit(0);
```

```
}
```

The only essential difference between example #6 above and example #1 (shown on page 236) is that the user's DN and the password are passed to ldap_simple_bind_s() rather than just NULL values. This is, however, not very secure because the password is sent over the network unencrypted and can be eavesdropped. A more secure way is to use SSL to encrypt the session traffic between client and server.

Example #7 below shows how to connect to an IBM SecureWay Directory using the SSL-related API calls. Prior to using SSL, it had to be configured properly (see 5.4, "Configuring SSL Security" on page 111).

As shown in the following, the example uses *server authentication* only although the IBM SecureWay Directory supports *client authentication* as well (see 5.4.3, "Configuring an LDAP Server to Use SSL" on page 116).

```
/*
* example #7
*/

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

main()
{
        LDAP            *ld;
        char            *User = "cn=Directory Manager";
        char            *Passwd = "1234qwer";
        char            *keyring = "/home/root/keys/venus-keyfile.kdb";
        char            *keyring_pw = NULL, *name = NULL;
        int             rc, ssl_rc;

/* initialize SSL client */
if ((rc = ldap_ssl_client_init(keyring, keyring_pw, 10, &ssl_rc)) != NULL){
                perror("ldap_ssl_client_init error");
                exit(1);
                }
else {
        printf("ldap_ssl_client_init done.\n");
}

/* initiate SSL session */
if ((ld = ldap_ssl_init("saturn.itso.austin.ibm.com", LDAPS_PORT, name)) ==
                          NULL) {
        perror("ldap_ssl_init error");
        exit(1);
```

```
        }
else {
        printf("Success: ldap_ssl_init\n");
}

/* optionally set SSL options using ldap_set_option() here */

/* authenticate to server */
if (ldap_simple_bind_s(ld, User, Passwd) != LDAP_SUCCESS) {
                ldap_perror(ld, "ldap_simple_bind_s");
                exit(1);
}
printf("Authenticated as %s\n", User);

/* ..............................
*   do something, for example
*   ask the server for information
* ............................... */

/* close and free connection resources */
ldap_unbind(ld);
exit(0);
}
```

A new call in example #7 above, `ldap_ssl_client_init()`, initializes the client's SSL library. It requires four parameters:

keyring — This specifies the name of the keyring file. It usually contains the certificates of the trusted (by the client) certificate authorities. It can also contain a public key and the associated certificate if client authentication is required.

keyring_pw — The password that protects the keyring file. It is set when the keyring file is created with the ikmgui tool. A NULL password, as in our example, is accepted, which causes the password to be taken from a stash file (see 5.4, "Configuring SSL Security" on page 111).

ssl_timeout — A timeout value, in seconds, for the SSL connection. It is set to 10 (seconds) in example #7.

ssl_rc — A return code that contains additional error conditions as defined in ldapssl.h.

The second difference to example #6 is that the ldap_init() call is replaced by ldap_ssl_init(), which initializes an SSL connection on port LDAPS_PORT instead of LDAP_PORT. This constant, LDAPS_PORT, is set in the ldap.h file

to 636, which is the default SSL port for LDAP. The actual SSL connection, however, is only established when the ldap_simple_bind_s() call takes place.

The optional ldap_set_option() call allows an application to set various SSL options, such as the ciphers to be used. This may be important if you want to restrict the level of security to only certain ciphers.

Note that the user's ID (DN) and the password are still being sent to the server (by `ldap_simple_bind_s()`), but since a secure SSL session has been established, they are encrypted and cannot be eavesdropped on the network.

A very general authentication command available with LDAP Version 3, which offers access to different authentication methods, is the `ldap_sasl_bind_s()` function (or its asynchronous version `ldap_sasl_bind()`). Its syntax is:

```
int ldap_sasl_bind_s(LDAP           *ld,
                     char           *dn,
                     char           *mechanism,
                     struct berval  *cred,
                     LDAPControl    **serverctrls,
                     LDAPControl    **clientctrls,
                     struct berval  **servercredp);
```

In principle, the SASL functions can be understood as a general authentication framework for the LDAP client. They take as arguments, among others, the DN (the name of the entry to bind as), the mechanism (authentication method), and the credentials used to authenticate. The format of the credentials passed to the SASL command depends on the mechanism used. If the special constant, LDAP_SASL_SIMPLE, is passed, then basic authentication is requested. This is equivalent to using ldap_simple_bind() (or ldap_simple_bind_s()). Other methods, such as Kerberos or S/Key, can be used as well.

SSL, or more general its successor TLS (Transport Layer Security), can get integrated within the SASL framework through its EXTERNAL mechanism. When this method is chosen and the cred field is empty, the server determines the client's identity through external information. This could be an SSL client certificate issued for the distinguished name used in the ldap_sasl_bind() function to bind to the LDAP server. If the cred field is not empty, the LDAP server has to verify that the client's authenticated TLS credentials allow use of the credentials passed to ldap_sasl_bind().

8.2.9 Multithreaded Applications

The LDAP Version 2 API only dealt with the single-threaded model of the LDAP API. In an effort to overcome such vendor-dependent approaches, the

LDAP Version 3 C-language API has been extended with a common set of multithreaded function calls.

As an example, let us assume that a large number of directory entries have to be modified. This can, of course, be done by sequentially stepping through the list of entries that have to be modified and by doing the changes one after the other. The list will have to be composed first, for example by performing an adequate search.

One way to accomplish the modifications is to divide the list of entries into, for example, 10 equal sized blocks. Then, multiple connections could be opened to the server (issuing `ldap_init()` multiple times), each one running in its own thread using the multithreading facilities of the operating system. Every thread would obtain its own connection handle for its connection. Every connection handle contains information about the connection itself, fields for error handling, and so on. This results in a multithreaded, multiconnection application, as depicted in Figure 65.

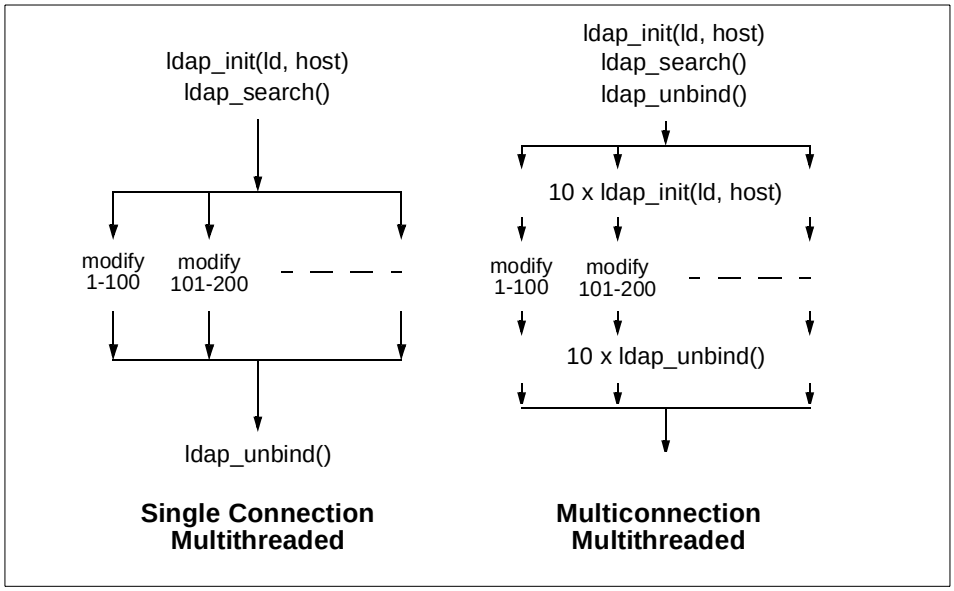


Figure 65. Multiple parallel threads

The single connection approach would be to use only one connection but several threads operating on it. Because one connection means only one connection handle, the LDAP Version 3 C API supports functions to isolate information that is vital for each thread protecting it from getting tampered with by functions from another thread. This is, for example, done by creating

thread-specific fields in the connection handle for error handling. Here, the error code of the last instruction issued within the attached thread is stored. Only functions within the same thread can access and evaluate it.

The IBM LDAP C client libraries are generally thread safe for all supported platforms (AIX, Solaris, OS/2, HP-UX, Win95, OS390, and WinNT). While a multithreaded application can safely use the LDAP library on multiple threads within the application, there are a few considerations to keep in mind.

- State information for a LDAP connection is maintained in the ld returned from the following LDAP APIs:
  - ldap_init()
  - ldap_ssl_init()
  - ldap_open() (deprecated)

  The ld, as returned from one of these APIs, is actually a pointer to a data structure that contains information relevant to the LDAP connection. The ld is maintained by the application and is supplied on subsequent LDAP API invocations.
- Using the LDAP connection (that is the ld) on the thread on which it was created (using ldap_init(), ldap_ssl_init() or ldap_open()) is a good model. This model avoids the possibility of conflicts that could arise if multiple threads attempt to concurrently process the results of an operation.
- An application may be designed to submit requests on a thread with results being fetched on a different thread. This is also a good model since it avoids the situation where two or more threads are attempting to concurrently fetch results associated with a single LDAP connection.
- The ldap_get_errno() API obtains information with respect to the most recent error that occurred for the specified LDAP connection. It does not necessarily return the most recent LDAP error that occurred on the thread on which it was issued. This could be a factor even if two threads are using the same ld in a well-coordinated fashion.
- In general, to avoid unpredictable results, it is recommended that the application be designed so that multiple threads are not concurrently using the same LDAP connection. An application following this guideline has a variety of models to choose from.
  - Using a master thread (or threads to create connections) and passing work and the connection to a thread in a pool of worker threads.
  - Dispatching work to a thread from a pool of threads where all work related to a connection is performed on the thread (such as ldap_init(), ldap_simple_bind(), or ldap_unbind()).

- Using one thread to issue an asynchronous request and using another thread to wait for and process results.

This brief discussion about multithreaded application programming in LDAP Version 3 concludes the introduction to building LDAP-enabled applications. There is, of course, a lot more to it, but it would be beyond the scope of this book to elaborate more on the complete LDAP API and LDAP programming techniques. We refer you to the online *Programming Reference* that comes with the product.

8.3 Special Programming Topics

This section discusses several design and implementation topics for LDAP-enabling your product using the IBM SecureWay Directory. Think of this section as applied or advanced LDAP programming. You will find some suggestions on how to avoid known traps in directory exploitation, some workarounds for current limitations of the LDAP architecture, and some strong direction (or rules) for data modeling, security, and deploying your LDAP-enabled application in a production environment.

8.3.1 Data Considerations, Discrete Attributes versus Blobs

Data can be stored in the directory in a variety of forms: Human readable attributes, binary (opaque) attributes, and structure attributes (combination of human readable and binary). As you need to define new attributes, you will need to decide on the format those attributes will take, that is, the syntax for each attribute. The current implementation supports a number of syntaxes, of which, the most useful are:

- Binary
- Boolean
- IA5 String (ces, case-sensitive string)
- Directory String (cis, case-insensitive string)
- DN (Distinguished Name)
- integer
- Telephone Number
- Generalized Time
- UTC Time

All the syntaxes, except binary, produce values of the attribute that are human readable. The binary syntax is opaque (think of it as similar to viewing object code).

A key to making the directory usable (across multiple exploiters) is to apply a correct syntax to an attribute. This means that there are appropriate uses of the binary syntax (also referred to as blobs) and inappropriate uses. In general, information placed in the directory service will be human readable. Exceptions to this are binary objects, such as certificates or JPEG photos.

Therefore, it is important that you examine closely (and understand) the data to be stored in the directory and define the needed object classes and attributes to express that information. Where objects and attributes match those currently defined in the schema, they should be used. If application configuration information is to be stored in a directory, it should not be placed in the directory as a single concatenated binary attribute (blob). The IBM SecureWay Directory access control functions should be used to control access to a directory entry rather than relying on making the entry private or unreadable by storing it as a blob.

Directory entries stored as blobs have a direct and negative impact on implementors' abilities to achieve the overall objectives for providing a centralized LDAP-based directory service. It is impossible to provide a converged consistent administrative model, impossible for applications to share information, and, therefore, impossible to obtain any reuse of object classes and attributes across applications inserting information in the directory.

8.3.2 Caching Considerations

The IBM SecureWay Directory uses server-side caching, via DB2's caching mechanisms, to improve lookup times on frequently accessed data. With appropriate tuning, sub-second response times for first time lookups and even faster for server-side cached entries are possible even with 15 million entries in the IBM SecureWay Directory.

If an application requires even faster response than this on frequently accessed data, then it may need to implement some form of client-side caching.

Determining whether to use client-side caching involves evaluating the tradeoff between improving performance and reducing consistency guarantees. When a lookup is done to the LDAP server, the result is guaranteed to be up-to-date regardless of whether the result came from the

database or from the server-side cache. But when a client decides to use locally cached data, it is possible that another client may have updated that entry since it was last retrieved.

Each application will have different consistency requirements. When a directory is used for naming, as in DNS or CDS, the client can tell whether the location information contained in an entry is stale by using it to contact the named entity. If the location information is correct, the connection attempt will succeed. If the information is incorrect (stale), the connection attempt will fail. At this point, the client can recover by repeating the name lookup with caching disabled using the dontUseCopy service control. So, naming information can be considered a hint and has relatively loose consistency requirements.

By contrast, an application that authenticates users using certificates may want to check for certificate revocation with caching disabled, or it may just require that revocation information be no more than one hour old. An e-mail client may be content to just require that e-mail addresses be less than a day old since users typically change addresses infrequently.

If an application can accept data with a given age, it may be appropriate to introduce a time-to-live (ttl) attribute. An Internet Draft, *A Simple Caching Scheme for LDAP and X.500 Directories*, <draft-ietf-asid-ldap-cache-01.txt>, has expired at the time of writing but described such a solution.

This draft proposed a simple caching model similar to that used by the DNS. The new attribute, ttl, is defined to specify the maximum time for which a cached copy of any attributes in the entry should be considered valid. The ttl attribute *should* be allowed in every entry that may be cached.

A new object class, cacheObject, could be defined, which allows an entry to have the new ttl attribute even if the server implementation does not support operational attributes.

The ttl attribute contains the time, for example, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy fetched. A value of 0 (zero) could imply that the object must not be cached.

So, an application that wishes to implement client-side caching using the ttl attribute could define a thin caching layer that wraps the LDAP queries it makes. The caching layer would keep a cache of returned entries along with a timestamp representing when each entry expires (fetch time + ttl).

When the application wishes to retrieve an entry, it would first check its cache for an unexpired entry, and if it found one, return the cached information. The application could also optionally pass in a parameter to override the ttl attribute if a particular query needed tighter (or looser) consistency constraints than were specified by the ttl attribute.

Appendix A. Standards

This appendix lists the pertinent formal and informal standards (RFCs and Internet Drafts) that relate to LDAP. A brief description or the actual abstract of the RFCs or the Internet Drafts, as it can be found on many Internet sites (for example, `http://www.ietf.org`), is added for your reference and convenience.

RFC 2222, Simple authentication and security layer (SASL)

This describes a method for adding authentication support to connection-based protocols. To use this specification, a protocol includes a command for identifying and authenticating a user to a server and for optionally negotiating protection of subsequent protocol interactions. If its use is negotiated, a security layer is inserted between the protocol and the connection. This document describes how a protocol specifies such a command, defines several mechanisms for use by the command, and defines the protocol used for carrying a negotiated security layer over the connection.

RFC 2251, Lightweight directory access protocol (v3)

This describes the LDAP Version 3 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. 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.

RFC 2252, Lightweight directory access protocol (v3): Attribute syntax definitions

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.

RFC 2253, Lightweight directory access protocol (v3): UTF-8 string representation of distinguished names

This RFC defines how distinguished names are represented as strings. A string representation is easy to encode and decode and is also human readable.

RFC 2254, The string representation of LDAP search filters

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=Diana*)) searches for entries that either have a surname attribute of Smith or that have a common name attribute that begins with Diana.

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.

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.

A simple LDAP change notification mechanism <draft-ietf-ldapext-psearch-01.txt>

This document defines two controls that extend the LDAPv3 [LDAP] search operation to provide a simple mechanism by which an LDAP client can receive notification of changes that occur in an LDAP server. The mechanism is designed to be very flexible yet easy for clients and servers to implement.

LDAPv3 security parameters <draft-hassler-ldapv3-secparam-00.txt>

This document defines an LDAP control called LDAPSecurityParameters for transferring security parameters with LDAP operations. With this control, it is possible to append digital signature to LDAP operations and, in this way, provide for message authenticity, message integrity, non-repudiation of message origin, and message freshness.

Authentication methods for LDAP <draft-ietf-ldapext-authmeth-03.txt>

This document specifies particular combinations of security mechanisms that are required and recommended in LDAP implementations.

LDAPv3 triggered search control <draft-ietf-ldapext-trigger-01.txt>

This document defines a LDAPv3 control to be used on the search request to allow a client to retrieve information on changes that are made to the directory information tree held by that server.

The LDAP data interchange format (LDIF) - Technical specification <draft-good-ldap-ldif-03.tx>

This document describes a file format suitable for describing directory information or modifications made to directory information. The file format, known as LDIF for LDAP Data Interchange Format, is typically used to import and export directory information between LDAP-based directory servers or to describe a set of changes that are to be applied to a directory.

Access control requirements for LDAP <draft-ietf-ldapext-acl-reqts-01.txt>

This document describes the fundamental requirements of an access control list (ACL) model for the Lightweight Directory Application Protocol (LDAP) directory service. It is intended to be a gathering place for access control requirements needed to provide authorized access to and interoperability between directories.

Lightweight directory access protocol (v3): Extensions for dynamic directory services <draft-ietf-asid-ldapv3-dynamic-08.txt>

This document defines the requirements for dynamic directory services and specifies the format of request and response extended operations for supporting client-server interoperation in a dynamic directories environment. The Lightweight Directory Access Protocol (LDAP) supports lightweight

access to static directory services allowing relatively fast search and update access. Static directory services store information about people that persists in its accuracy and value.

The Java LDAP application program interface <draft-ietf-ldapext-ldap-java-api-03.txt>

This document defines a java language application program interface to the lightweight directory access protocol (LDAP) in the form of a class library. It complements but does not replace RFC 1823, which describes a C language application program interface. It updates and replaces Draft draft-ietf-ldapext-ldap-java-api-01.txt in adding minor API extensions.

Appendix B. Special Notices

This publication is intended to help I/T architects and system engineers to design and implement a centralized directory based on the LDAP standard. The information in this publication is not intended as the specification of any programming interfaces that are provided by the IBM SecureWay Directory. See the PUBLICATIONS section of the respective IBM Programming Announcement for the IBM SecureWay Directory, or the IBM product or platform-specific announcements (IBM OS/390, IBM OS/400, IBM AIX, Sun Solaris, or Microsoft Windows NT) for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer

responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX ®
DB2 ®
IBM ®
OS/400 ®
S/390 ®
WebSphere
AS/400 ®
eNetwork
OS/390 ®
RS/6000 ®
SecureWay ®

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

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

MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC. (For further information, see http//www.setco.org/aboutmark.html)

Other company, product, and service names may be trademarks or service marks of others.

Appendix C. Other References and Related Publications

The references and publications listed in this appendix are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 283.

- *Understanding LDAP*, SG24-4986
- *Ready for e-business: OS/390 Security Server Enhancements*, SG24-5158

C.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs:

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbook	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

C.3 Other Publications

These publications are also relevant as further information sources:

- *OS/390 V2R7 Security Server LDAP Server Administration and Usage Guide*, SC24-5861
- *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, ISBN 1-57870-000-0
- *X.500 Directory Services; Technology and Deployment*, ISBN 1-85032-879-X

- *LDAP Version 3: The Maturing of the Internet Directory Standard*, The Burton Group, 1998
- *The Advent of Directory-Enabled Computing v2*, The Burton Group, 1995
- *Directory-Enabled Networks Initiative*, The Burton Group, 1997

C.4 The Internet Engineering Task Force (IETF)

The Internet Engineering Task Force (IETF) is an open international community to design and discuss future Internet technologies. The people belonging to this group are network designers, operators, and researchers from commercial and non-commercial organizations. The task is to design open standards for common use in the Internet. The group is open to any interested individual.

The actual work of the IETF is done in workgroups, which are organized by topics into several areas, for example routing, transport, security, and so on. The workgroups are grouped into areas and managed by area directors. These area directors are members of the Internet Engineering Steering Group (IESG). Providing architectural oversight is the Internet Architecture Board (IAB). Both the IESG and IAB are chartered by the Internet Society (ISOC) for these purposes. The general area director also serves as the chair of IETF and IESG and is an ex-official member of the IAB.

The central coordinator is the Internet Assigned Numbers Authority (IANA), which coordinates the unique assignment of parameter values for Internet protocols. The IANA is chartered by the ISOC to act as the clearinghouse to assign and coordinate the use of numerous Internet protocol parameters.

New technologies are invented and discussed in so-called IETF drafts (or Internet Drafts). These drafts and the basic design ideas are posted to the mailing list and are discussed until general consensus is reached to stop work on the draft or progress it to RFC by requesting approval of the area director and the IESG. At approval, an RFC number is assigned. The RFC is the base description for new versions and enhancements.

The IETF Web site can be reached at:

```
http://www.ietf.org
```

On this Web site, there are links to IETF Workgroups, Internet Drafts, mailing lists, and so on.

C.5 The University of Michigan (UMICH)

The University of Michigan was, and still is, an important contributor in the development of LDAP and can be considered a reliable, neutral source for extensive information and program source code for LDAP servers and clients.

The UMICH's home page is at:

```
http://www.umich.edu
```

The UMICH's LDAP page can be accessed at:

```
http://www.umich.edu/~dirsvcs/ldap
```

This latter page contains, among others, links to online LDAP documentation from the UMICH and others and downloadable software, most of which is source code.

C.6 IBM Internet Wet Site for the IBM SecureWay Directory

The IBM Web site for LDAP and related directory topics is:

```
http://www.ibm.com/software/enetwork/directory
```

C.7 IBM Intranet Web Site

The first point of contact for IBM development teams is the IBM eNetwork LDAP Directory Intranet Web site at:

```
http://w3.ibm.com/software/ldap
```

The Web site includes:

- IBM eNetwork LDAP Directory Code for Download
- GA Level - IBM eNetwork LDAP Directory for all available platforms
- Beta level - IBM eNetwork LDAP Directory (when available)
- LDAP and JNDI Client code for all available platforms
- Online Documents
- IBM eNetwork LDAP Directory Exploitation Guide (this document)
- IBM eNetwork LDAP Directory Product Documentation
- Current Schema and Namespace Definitions
- Library for Schema and Namespace Extensions from Products and Standards Bodies

- Current Requirements from Exploitation Projects
- Links to IETF and other interesting LDAP and related Web Sites
- A central e-mail ID for submitting questions and problems to the LDAP developmen team: `ldap@us.ibm.com`, or via Lotus Notes: `ldap/Austin/IBM`

C.8 Lotus Notes Discussion Database

In addition to the information and code available via the LDAP Web site, the DSS Architecture Team has a Lotus Notes-based discussion database for discussing directory exploitation issues with the growing number of IBM teams working with LDAP directories.

To access the database manually from Lotus Notes:

- File-Database-Open from the menu bar to get the *Open Database* dialog box.
- In the Server field, type: `dss.austin.ibm.com`
- In the Filename field, type: `dss\direxp.nsf`
- Click the **Open** button

Note that you might have to add a connection document for this server.

C.9 Software Development Kits

Below, you find some Web siteswhere you can download SDKs offered by different vendors for a wide variety of platforms. You might find some useful information, such as documentation and FAQ (Frequently Asked Questions) lists and links to other interesting LDAP-related places, there as well.

The University of Michigan's LDAP server code, a C language SDK, and other links to documentation and LDAP mailing lists can be found at the following link:

```
http://www.umich.edu/~dirsvcs/ldap/
```

IBMs C and Java SDKs can be found at:

```
http://www.ibm.com/java
```

Netscape offers a C and a Java SDKs at:

```
http://developer.netscape.com/software/sdks/index.html
```

C.10 Other Sources

The following are links to various Web sites that carry information related to the subject of this book.

C.10.1 LDAP, General

For general information about LDAP, please refer to the Web sites of the IETF and the University of Michigan as listed earlier in this appendix. If you search the Web for LDAP, there will be thousands of hits. The following are just a few links to related Web sites that you might find interesting.

An LDAP Roadmap and FAQs:

```
http://www.kingsmountain.com/ldapRoadmap.shtml
http://www.critical-angle.com/ldapworld/ldapfaq.html
```

IBM eNetwork Security and Directory library:

```
http://www.ibm.com/software/security/firstsecure/library
```

Critical Angle Inc. (Innosoft International Inc.) hosts a series of interesting information and links at:

```
http://www.critical-angle.com/ldapworld/index.html
```

A System Administrator's view of LDAP:

```
http://people.netscape.com/bjm/whyLDAP.html
```

C.10.2 Request for Comments (RFCs) and other References

A good source for accessing RFCs with search capabilities, besides the IETF Web site (see above), is provided by the Information Sciences Institute (ISI) at:

```
http://www.rfc-editor.org
```

This ISI Web site includes a number of links to other RFC sources. If, for some reason, you cannot access the link above, try one of the following:

```
http://www.pasteur.fr/other/computer/RFC
http://www.garlic.com/~lynn/rfcietf.html
http://www.nexor.com/public/rfc/index/rfc.html
http://www.csl.sony.co.jp/rfc
```

Unicode descriptions, including tables and graphics of all characters, can be found at:

```
http://www.unicode.org
```

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** `http://www.redbooks.ibm.com/`

  Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at `http://www.redbooks.ibm.com/` and for IBM employees at `http://w3.itso.ibm.com/`.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at `http://w3.itso.ibm.com/` and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook. residency, and workshop announcements at `http://inews.ibm.com/`.

IBM Redbook Fax Order Form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

ACL	Access Control List
APS	Auxiliary Storage Pool
ASN	Abstract Syntax Notation
ASP	Auxiliary Storage Pool
BLOB	Binary Large Objects
BNF	Backus-Naur Form (also Backus Normal Form)
CA	Certificate Authority
CCITT	Comite Consultatif International Telephonique et Telegraphique
CDS	Cell Directory Service (DCE)
CDSA	Common Data Security Architecture
CIM	Common Information Model
CRAM-MD5	Challenge-Response Authentication Mechanism - Message Digest 5
DAP	Directory Access Protocol (X.500)
DARPA	Defense Advanced Research Projects Agency
DAS	Directory Assistance Service
DB2	Database 2
DCE	Distributed Computing Environment
DEN	Directory Enabled Networks
DES	Data Encryption Standard
DIT	Directory Information Tree
DMTF	Desktop Management Task Force
DN	Distinguished Name
DNS	Domain Name System
DSA	Directory Services Agent (synonym for Directory Server)
DSE	DSA-Specific Entry
DSS	Directory & Security Services
DTS	Distributed Time Service
EDI	Electronic Data Interchange
EJB	Enterprise Java Beans
FTP	File Transfer Protocol
GDA	Global Directory Agent
GDS	Global Directory Service
GSO	Global Sign-On
GSSAPI	Generic Security Service API
HTTP	Hypertext Transport Protocol
IAB	Internet Architecuture Board
IANA	Internet Assigned Numbers Authority
IBM	International Business Machines Corporation
IETF	Internet Engineering Task Force
IESG	Internet Engineering Steering Group
IMAP	Internet Message Access Protocol
ISI	Information Sciences Institute
ISO	International Standards Organization
ISOC	Internet Society

ITSO	International Technical Support Organization
ITU-T	International Telecommunications Union - Telecommunications
JCL	Job Control Language
JDAP	Java Directory Access Protocol (context: Java LDAP Application Programming Interface)
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface (Sun)
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
LIPS	Lightweight Internet Person Schema
MIME	Multipurpose Internet Mail Extensions
NDS	Novell Directory Services
NOS	Network Operating System
NSI	Name Service Interface (DCE)
OSF	Open Software Foundation
OSI	Open Systems Interconnection
RDN	Relative Distinguished Name
RFC	Request for Comment
RPC	Remote Procedure Call
SASL	Simple Authentication and Security Layer
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SPI	Service Provider Interface
SPUFI	SQL Processor Using File Input
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TME	Tivoli Management Environment
TMR	Tivoli Management Region
UCS	Unicode Character Set
UMICH	University of Michigan
URL	Uniform Resource Locator
USS	UNIX Systems Services
UTF	UCS Tranformation Format
WAN	Wide Area Network

Index

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

Y

ITSO Redbook Evaluation

LDAP Implementation Cookbook
SG24-5110-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

SG24-5110-00
Printed in the U.S.A.

LDAP Implementation Cookbook

SG24-5110-00