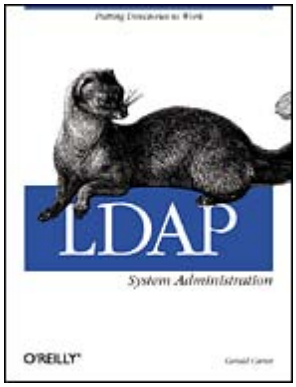
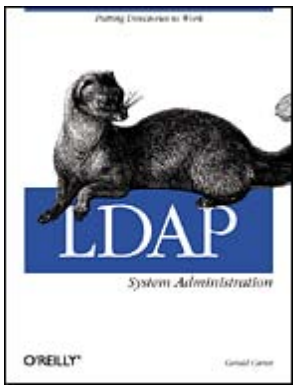


LDAP System Administration

By Gerald Carter

Publisher : O'Reilly
Pub Date : March 2003
ISBN : 1-56592-491-6
Pages : 308

If you want to be a master of your domain, LDAP System Administration will help you get up and running quickly regardless of which LDAP version you use. After reading this book, even with no previous LDAP experience, you'll be able to integrate a directory server into essential network services such as mail, DNS, HTTP, and SMB/CIFS.



LDAP System Administration

By Gerald Carter

Publisher : O'Reilly
Pub Date : March 2003
ISBN : 1-56592-491-6
Pages : 308

Copyright

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Preface

In 1999 I began experimenting with the Lightweight Directory Access Protocol (LDAP) and immediately became frustrated by lack of documentation. I set out to write the book that I needed, and I believe that I accomplished that goal. After teaching instructional courses on LDAP for the past few years, I have come to the belief that many people share the same frustration I felt at the beginning of my LDAP career. Managers and administrators alike can sometimes be dazzled (or disgusted) by the plethora of acronyms in the IT industry. The goal of this book is to cut through the glossy vendor brochures and give you the knowledge and tools necessary to deploy a working directory on your network complete with integrated client applications.

Directory services have been a part of networks in one way or another for a long time. LDAP directories have been growing roots in networks for as long as people have been proclaiming the current year to be the "year of LDAP." With increasing support from vendors in the form of clients and servers, LDAP has already become a staple for many networks. Because of this gradual but steady growth, people waiting for the LDAP big bang may be disappointed. You may wake up one morning and find that one of your colleagues has already deployed an LDAP-based directory service. If so, this book will help you understand how you can use the services that LDAP provides. If you are at the beginning of a project, this book will help you focus on the important points that are necessary to succeed.

How This Book Is Organized

This book is divided into two sections of five chapters each and a section of appendixes. You will most likely get the most out of this book if you implement the example directories as they are covered. With only a few exceptions, all client and server applications presented here are freely available or in common use.

Part I : LDAP Basics

Part I focuses on getting acquainted with LDAP and with the OpenLDAP server. In this part, I answer questions such as: "What is lightweight about LDAP?," "What security mechanisms does LDAP support for preventing unauthorized access to data?," and "How can I build a fault-tolerant directory service?" In addition, the first part of the book helps you gain practical experience with your own directory using the community-developed and freely available OpenLDAP server.

Chapter 1 is a high-level overview of directory services and LDAP in particular.

Chapter 2 digs into the details of the Lightweight Directory Access Protocol.

Chapter 3 uses the free server distribution from OpenLDAP.org as an example to present practical experience with an LDAP directory.

Chapter 4 provides some hands-on experience adding, modifying, and deleting information from a working directory service.

Chapter 5 wraps up the loose ends of some of the more advanced LDAPv3 and OpenLDAP features.

Part II : Application Integration

Part II is all about implementation. Rather than present an LDAP cookbook, I bring different applications together in such a way that information common to one or more clients can be shared via the directory. You will see how to use LDAP as a practical data store for items such as user and group accounts, host information, general contact information, and application configurations. I also discuss integration with other directory services such as Microsoft's Active Directory, and how to develop your own Perl scripts to manage your directory service.

Chapter 6 explains how an LDAP directory can be used to replace Sun's Network Information Service (NIS) as the means to distribute user and group accounts, host information, automount maps, and other system files.

Chapter 7 presents information related to both mail clients (Eudora, Mozilla, Outlook, and Pine) and servers (Sendmail, Postfix, and Exim).

Chapter 8 explains how to use an LDAP directory to share information among essential network services such as FTP, HTTP, LPD, RADIUS, DNS, and Samba.

Conventions Used in This Book

The following conventions are used in this book:

Italic

Used for file, directory, user, and group names. It is also used for URLs and to emphasize new terms and concepts when they are introduced.

Constant Width

Used for code examples, system output, parameters, directives, and attributes.

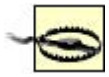
Constant Width Italic

Used in examples for variable input or output (e.g., a filename).

`Constant Width Bold`

Used in code examples for user input and for emphasis.

This icon designates a note, which is an important aside to the nearby text.



This icon designates a warning relating to the nearby text.

Comments and Questions

We at O'Reilly have tested and verified the information in this book to the best of our abilities, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:
O'Reilly & Associates, Inc. 1005 Gravenstein Highway North Sebastopol, CA 95472 (800) 998-9938 (U.S. and Canada) (707) 827-7000 (international/local) (707) 829-0104 (fax)

You can also contact O'Reilly by email. To be put on the mailing list or request a catalog, send a message to:
info@oreilly.com

We have a web page for this book, which lists errata, examples, and any additional information. You can access this page at:
http://www.oreilly.com/catalog/ldapsa/

To comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com

For more information about O'Reilly books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:
http://www.oreilly.com/

Acknowledgments

At the end of every project, I am acutely aware that I could never have reached the end without the grace provided to me by God through my Savior, Jesus Christ. I hope He is proud of how I have spent my time. I am also very conscious of the patience bestowed upon me by my wife, Kristi, who is always there to listen when I need to talk and laugh when I need a smile. Thank you.

There is a long list of people who have helped make this book possible. I do not claim that this is a complete list. Mike Loukides has shown almost as much patience as my wife waiting on this book to be completed. I am in great debt to the technical reviewers who each provided comments on some version of this manuscript: Robbie Allen, David Blank-Edelman, Æleen Frisch, Robert Haskins, Luke Howard, Scott McDaniel, and Kurt Zeilenga. Thanks to Æleen for convincing me to do this (even if I complained more than once). I must also mention the various coffee shops, particularly the Books-A-Million in Auburn, AL, that have allowed me to consume far more than my fair share of caffeine and electricity.

Finally, a huge amount of recognition must be given to the developers who made various pieces of software available under open source and free software licenses. It is such an enjoyable experience to be able to send and receive feedback on problems, bugs, and solutions. Any other way would just be too painful.

Part I: LDAP Basics

Chapter 1. "Now where did I put that...?", or "What is a directory?"

I have a fairly good memory for numbers, phone numbers in particular. This fact amazes my wife. For those numbers I cannot recall to the exact digit, I have a dozen or so slots in my cell phone. However, as the company I worked for grew, so did the list of people with whom I needed to stay in contact. And I didn't just need phone numbers; I needed email and postal addresses as well. My cell phone's limited capabilities were no longer adequate for maintaining the necessary information.

So I eventually broke down and purchased a PDA. I was then able to store contact information for thousands of people. Still, two or three times a day I found myself searching the company's contact database for someone's number or address. And I still had to go to other databases (phone books, corporate client lists, and so on) when I needed to look up someone who worked for a different company.

Computer systems have exactly the same problem as humans—both require the capability to locate certain types of information easily, efficiently, and quickly. During the early days of the ARPAnet, a listing of the small community of hosts could be maintained by a central authority—SRI's Network Information Center (NIC). As TCP/IP became more widespread and more hosts were added to the ARPAnet, maintaining a centralized list of hosts became a pipe dream. New hosts were added to the network before everyone had even received the last, now outdated, copy of the famous *HOSTS.TXT* file. The only solution was to distribute the management of the host namespace. Thus began the Domain Name System (DNS), one of the most successful directory services ever implemented on the Internet.[1]

[1] For more information on the Domain Name System and its roots, see DNS and BIND, by Paul Albitz and Cricket Liu (O'Reilly).

DNS is a good starting point for our overview of directory services. The global DNS shares many characteristics with a directory service. While directory services can take on many different forms, the following five characteristics hold true (at a minimum):

- A directory service is highly optimized for reads. While this is not a restriction on the DNS model, for performance reasons many DNS servers cache the entire zone information in memory. Adding, modifying, or deleting an entry forces the server to reparse the zone files. Obviously, this is much more expensive than a simple DNS query.
- A directory service implements a distributed model for storing information. DNS is managed by thousands of local administrators and is connected by root name servers managed by the InterNIC.
- A directory service can extend the types of information it stores. Recent RFCs, such as RFC 2782, have extended the types of DNS records to include such things as server resource records (RRs).
-

1.1 The Lightweight Directory Access Protocol

Of course, you didn't buy this book to read about the Domain Name System. And it's not likely that you were looking for a general discussion of directory services. This book is about a particular kind of directory service—namely, a service for directories that implement the Lightweight Directory Access Protocol (LDAP). LDAP has become somewhat of a buzzword in contemporary IT shops. If you are like me, sometimes you just have to ask, "Why all the fuss?" The fuss is not so much about LDAP itself, but about the potential of LDAP to consolidate existing services into a single directory that can be accessed by LDAP clients from various vendors. These clients can be web browsers, email clients, mail servers, or any one of a myriad of other applications.

By consolidating information into a single directory, you are not simply pouring the contents of your multitude of smaller pots into a larger pot. By organizing your information well and thinking carefully about the common information needed by client applications, you can reduce data redundancy in your directories and therefore reduce the administrative overhead needed to maintain that data. Think about all the directory services that run on your network and consider how much information is duplicated. Perhaps hosts on your network use a DHCP server. This server has a certain amount of information about IP addresses, Ethernet addresses, hostnames, network topology, and so forth in its configuration files. Which other applications use the same or similar information and could share it if it were stored in a directory server? DNS comes immediately to mind, as does NIS. If you have networked printers as well, think about the amount of information that's replicated on each client of the printing system (for example, */etc/printcap* files).

Now consider the applications that use your user account information. The first ones that probably come to mind are authentication services: users need to type usernames and passwords to log in. Your mail server probably uses the same username information for mail routing, as well as for services such as mailing lists. There may also be online phone books that keep track of names, addresses, and phone numbers, as well as personnel systems that keep track of job classifications and pay scales.

Imagine the administrative savings that would result if all the redundant data on your network could be consolidated in a single location. What would it take to delete a user account? We all know what that takes now: you delete the user from */etc/passwd*, remove him by hand from any mailing lists, remove him from the company phone list, and so on. If you're clever, you've probably written a script or two to automate the process, but you're still manipulating the same information that's stored in several different places. What if there was a single directory that was the repository for all this information, and deleting a user was simply a matter of removing some records from this directory? Life would become much simpler. Likewise, what would it take to track host-related information? What would it be worth to you if you could minimize the possibility that machines and users use out-of-date information?

This sounds like a network administrator's utopia. However, I believe that as more and more client applications use LDAP directories, making an investment in setting up an LDAP server will have a huge payoff long-term. Realistically, we're not headed for a utopia. We're going to be responsible for more servers and more services, running on more platforms. The dividends of our LDAP investment come when we significantly reduce the number of directory technologies that we have to understand and administer. That is our goal.

1.2 What Is LDAP?

The best place to begin when explaining LDAP is to examine how it got its name. Let's start at the beginning. The latest incarnation of LDAP (Version 3) is defined in a set of nine documents outlined in RFC 3377. This list includes:
RFC 2251-2256

The original core set of LDAPv3 RFCs
RFC 2829

"Authentication Methods for LDAP"
RFC 2830

"Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security"
RFC 3377

"Lightweight Directory Access Protocol (v3): Technical Specification"

1.2.1 Lightweight

Why is LDAP considered lightweight? Lightweight compared to what? (As we look at LDAP in more detail, you'll certainly be asking how something this complex could ever be considered lightweight.) To answer these questions, it is necessary to look at LDAP's origins. The roots of LDAP are closely tied to the X.500 directory service; LDAP was originally designed as a lighter desktop protocol used to gateway requests to X.500 servers. X.500 is actually a set of standards; anything approaching thorough coverage of X.500 is beyond the scope of this book.[2]

[2] Understanding X.500—The Directory, by David W. Chadwick, provides a good explanation of X.500 directories. While the book itself is out of print, an HTML version of it can be accessed from http://www.salford.ac.uk/its024/X500.htm.

X.500 earned the title "heavyweight." It required the client and server to communicate using the Open Systems Interface (OSI) protocol stack. This seven-layered stack was a good academic exercise in designing a network protocol suite, but when compared to the TCP/IP protocol suite, it is akin to traveling the European train system with four fully loaded footlockers.[3]

[3] For a quick, general comparison of the OSI model and the TCP/IP protocol stack, see Computer Networks, by Andrew S. Tannenbaum (Prentice Hall).

LDAP is lightweight in comparison because it uses low overhead messages that are mapped directly onto the TCP layer (port 389 is the default) of the TCP/IP protocol stack.[4] Because X.500 was an application layer protocol (in terms of the OSI model), it carried far more baggage, as network headers were wrapped around the packet at each layer before it was finally transmitted on the network (see Figure 1-1).

[4] A connectionless version of LDAP that provided access via UDP was defined by an Internet-Draft produced by the LDAP Extension Working Group of the IETF. However, the current draft expired in November, 2001. You can access the group's web site at http://www.ietf.org/html.charters/ldapext-charter.html.

Figure 1-1. X.500 over OSI versus LDAP over TCP/IP

1.3 LDAP Models

LDAP models represent the services provided by a server, as seen by a client. They are abstract models that describe the various facets of an LDAP directory. RFC 2251 divides an LDAP directory into two components: the protocol model and the data model. However, in Understanding and Deploying LDAP Directory Services, by Timothy A. Howes, Mark C. Smith, and Gordon S. Good (MacMillan), four models are defined:

Information model

The information model provides the structures and data types necessary for building an LDAP directory tree. An entry is the basic unit in an LDAP directory. You can visualize an entry as either an interior or exterior node in the Directory Information Tree (DIT). An entry contains information about an instance of one or more objectClasses. These objectClasses have certain required or optional attributes. Attribute types have defined encoding and matching rules that govern such things as the type of data the attribute can hold and how to compare this data during a search. This information model will be covered extensively in the next chapter when we examine LDAP schema.

Naming model

The naming model defines how entries and data in the DIT are uniquely referenced. Each entry has an attribute that is unique among all siblings of a single parent. This unique attribute is called the relative distinguished name (RDN). You can uniquely identify any entry within a directory by following the RDNs of all the entries in the path from the desired node to the root of the tree. This string created by combining RDNs to form a unique name is called the node's distinguished name (DN).

In Figure 1-4, the directory entry outlined in the dashed square has an RDN of cn=gerald carter. Note that the attribute name as well as the value are included in the RDN. The DN for this node would be cn=gerald carter,ou=people, dc=plainjoe,dc=org.

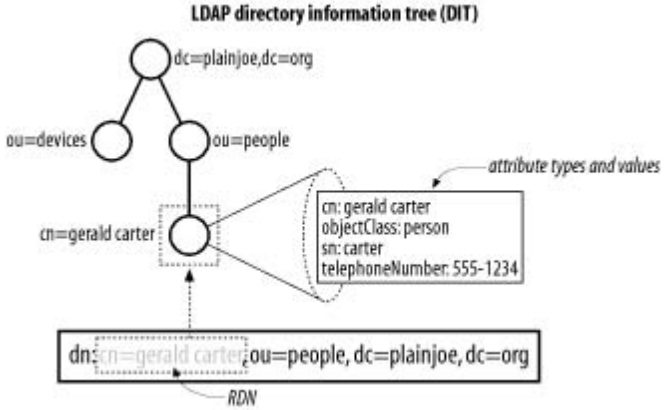
Functional model

The functional model is the LDAP protocol itself. This protocol provides the means for accessing the data in the directory tree. Access is implemented by authentication operations (bindings), query operations (searches and reads), and update operations (writes).

Security model

The security model provides a mechanism for clients to prove their identity (authentication) and for the server to control an authenticated client's access to data (authorization). LDAPv3 provides several authentication methods not available in previous protocol versions. Some features, such as access control lists, have not been standardized yet, leaving vendors to their own devices.

Figure 1-4. Example LDAP directory tree



At this high level, LDAP is relatively simple. It is a protocol for building highly distributed directories. In the next chapter, we will examine certain LDAP concepts such as schemas, referrals, and replication in much more depth.

Chapter 2. LDAPv3 Overview

Chapter 1 should have helped you understand the characteristics of a directory in general, and an LDAP directory in particular. If you still feel a little uncomfortable about LDAP, relax. This chapter is designed to flesh out some of the details that we glossed over. Your immediate goal should be to understand the basic building blocks of any LDAPv3 directory server. In the next chapter, we will start building an LDAP directory.

2.1 LDIF

Most system administrators prefer to use plain-text files for server configuration information, as opposed to some binary store of bits. It is more comfortable to deal with data in vi, Emacs, or notepad than to dig though raw bits and bytes. Therefore, it seems fitting to begin an exploration of LDAP internals with a discussion of representing directory data in text form.

The LDAP Interchange Format (LDIF), defined in RFC 2849, is a standard text file format for storing LDAP configuration information and directory contents. In its most basic form, an LDIF file is:

- A collection of entries separated from each other by blank lines
- A mapping of attribute names to values
- A collection of directives that instruct the parser how to process the information

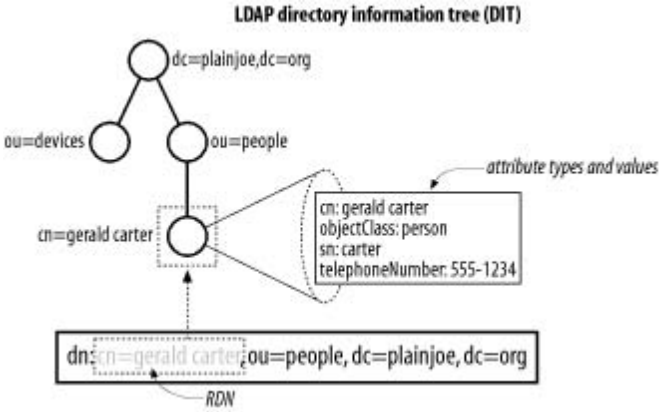
The first two characteristics provide exactly what is needed to describe the contents of an LDAP directory. We'll return to the third characteristic when we discuss modifying the information in the directory in Chapter 4.

LDIF files are often used to import new data into your directory or make changes to existing data. The data in the LDIF file must obey the schema rules of your LDAP directory. You can think of the schema as a data definition for your directory. Every item that is added or changed in the directory is checked against the schema for correctness. A *schema violation* occurs if the data does not correspond to the existing rules.

Figure 2-1 shows a simple directory information tree. Each entry in the directory is represented by an entry in the LDIF file. Let's begin with the topmost entry in the tree labeled with the distinguished name (DN) dc=plainjoe,dc=org:

```
# LDIF listing for the entry dn: dc=plainjoe,dc=org
dn: dc=plainjoe,dc=org
objectClass: domain
dc: plainjoe
```

Figure 2-1. An LDAP directory tree



We can make a few observations about LDIF syntax on the basis of this short listing:

2.2 What Is an Attribute?

The concepts of attribute types and attribute syntax were mentioned briefly in the previous chapter. Attribute types and the associated syntax rules are similar to variable and data type declarations found in many programming languages. The comparison is not that big of a stretch. Attributes are used to hold values. Variables in programs perform a similar task—they store information.

When a variable is declared in a program, it is defined to be of a certain data type. This data type specifies what type of information can be stored in the variable, along with certain other rules, such as how to compare the variable's value to the data stored in another variable of the same type. For example, declaring a 16-bit integer variable in a program and then assigning it a value of 1,000,000 would make no sense (the maximum value represented by a signed 16-bit integer is 32,767). The data type of a 16-bit integer determines what data can be stored. The data type also determines how values of like type can be compared. Is 3 < 5? Yes, of course it is. How do you know? Because there exists a set of rules for comparing integers with other integers. The syntax of LDAP attribute types performs a similar function as the data type in these examples.

Unlike variables, however, LDAP attributes can be multivalued. Most procedural programming languages today enforce "store and replace" semantics of variable assignment, and so my analogy falls apart. That is, when you assign a new value to a variable, its old value is replaced. As you'll see, this isn't true for LDAP; assigning a new value to an attribute adds the value to the list of values the attribute already has. Here's the LDIF listing for the ou=devices,dc=plainjoe,dc=org entry from Figure 2-1; it demonstrates the purpose of multivalued attributes:

```
# LDIF listing for dn: ou=devices,dc=plainjoe,dc=org
dn: ou=devices,dc=plainjoe,dc=org
objectclass: organizationalUnit
ou: devices
telephoneNumber: +1 256 555-5446
telephoneNumber: +1 256 555-5447
description: Container for all network enabled
 devices existing within the plainjoe.org domain
```

Note that the description attribute spans two lines. Line continuation in LDIF is implemented by leaving exactly one space at the beginning of a line. LDIF does not require a backslash (\) to continue one line to the next, as is common in many Unix configuration files.

The LDIF file lists two values for the telephoneNumber attribute. In real life, it's common for an entity to be reachable via two or more phone numbers. Be aware that some attributes can contain only a single value at any given time. Whether an attribute is single- or multivalued depends on the attribute's definition in the server's schema. Examples of single-valued attributes include an entry's country (c), displayable name (displayName), or a user's Unix numeric ID (uidNumber).

2.2.1 Attribute Syntax

An attribute type's definition lays the groundwork for answers to questions such as, "What type of values can be stored in this attribute?", "Can these two values be compared?", and, if so, "How should the comparison take place?"

Continuing with our telephoneNumber example, suppose you search the directory for the person who owns the phone number 555-5446. This may seem easy when you first think about it. However, RFC 2252 explains that a

2.3 What Is the dc Attribute?

Returning to our discussion of the topmost entry in Figure 2-1, we can now explain the meaning of the domain object class and the dc attribute. Here is the original LDIF listing for the entry:

```
# LDIF listing for the entry dn: dc=plainjoe,dc=org
dn: dc=plainjoe,dc=org
objectclass: domain
dc: plainjoe
```

The original recommendation for dividing the X.500 namespace was based on geographic and national regions. You frequently see this convention in LDAP directories as well, given the heritage that LDAP shares with X.500. For example, under X.500, the distinguished name for a directory server in the *plainjoe.org* domain might be:

```
dn: o=plainjoe,l=AL,c=US
```

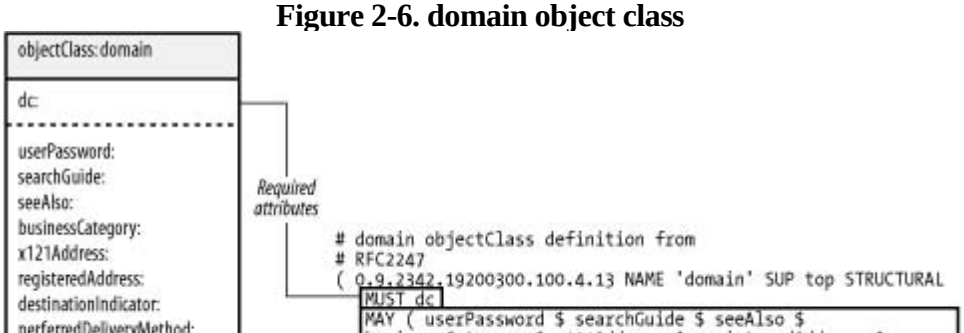
Here, the o attribute is the organizationName, the l attribute is the locality of the organization, and the c attribute represents the country in which the organization exists. However, there is no central means of registering such names, and therefore no general way to refer to the naming context of a directory server. RFC 2247 introduced a system by which LDAP directory naming contexts can be piggybacked on top of an organization's existing DNS infrastructure. Because DNS domain names are guaranteed to be unique across the Internet and can be located easily, mapping an organization's domain name to an LDAP DN provides a simple way of determining the base suffix served by a directory and ensures that the naming context will be globally unique.

A directory's naming context is the DN of its topmost entry. The naming context of the directory in our examples is dc=plainjoe,dc=org. This context is used by the LDAP server to determine whether it will be able to service a client request. For example, our directory server will return an error (or possibly a referral) to a client who attempts to look up the information in an entry named cn=gerald carter,ou=people,dc=taco,dc=org because the entry would be outside our naming context.

However, the server would search the directory (and return no information) if the client attempts to look up cn=gerald carter,ou=people,dc=taco,dc=plainjoe,dc=org. In this case, the directory's naming context does match the rightmost substring of the requested entry's DN. The server just does not have any information on the entry.

To support a mapping between a DNS domain name and an LDAP directory namespace, RFC 2247 defines two objects, shown in Figures 2-6 and 2-7, for storing domain components. The dcObject is an auxiliary class to augment an existing entry containing organizational information (e.g., an organizationalUnit). The domain object class acts as a standalone container for both the organizational information and the domain name component (i.e., the dc attribute). The organizationalUnit and domain objects have many common attributes.

Figure 2-6. domain object class



2.4 Schema References

One of the most frequent questions asked by newly designated LDAP administrators is, "What do all of these abbreviations mean?" Of course, the question refers to things such as cn, c, and sn. There is no single source of information describing all possible LDAPv3 attribute types and object classes, but there are a handful of online sites that can be consulted to cover the most common schema items:

RFC 3377 and related LDAPv3 standards (http://www.rfc-editor.org/)

The documents outlined in RFC 3377 provide a list of references for researching related LDAPv3 and X.500 topics. RFC 2256 in particular describes a set of X.500 schema items used with LDAPv3 directory servers.

LDAP Schema Viewer (http://ldap.akbkhome.com/)

This site, maintained by Alan Knowles, provides a nice means of browsing descriptions and dependencies among common LDAP attributes, object classes, and OIDs.

Object Identifiers Registry (http://www.alvestrand.no/objectid/)

This site can be helpful in tracking down the owner of specific OID arcs.

Sun Microsystems Product Documentation (http://docs.sun.com)

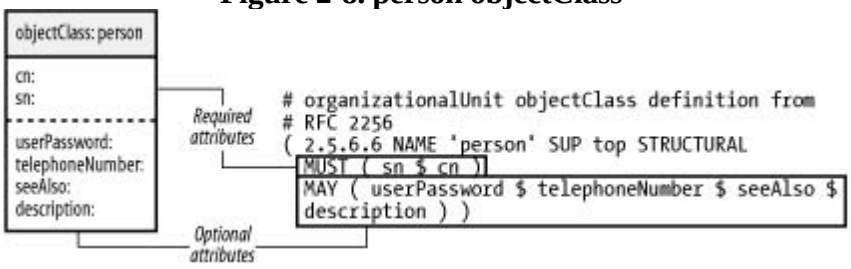
The SunOne Directory Server, formerly owned by Netscape Communications, includes a large set of reference documentation on various LDAP schema items. Even if you are not using the SunOne DS product, the schema reference can be helpful in understanding the meaning of various LDAP acronyms. Search the site for "LDAP schema reference" to locate the most recent versions of the product documentation.

2.5 Authentication

Why is authentication needed in an LDAP directory? Remember that LDAP is a connection-oriented, message-based protocol. The authentication process is used to establish the client's privileges for each session. All searches, queries, etc. are controlled by the authorization level of the authenticated user.

Figure 2-8 describes the person object class and gives you an idea of what other attributes are available for the cn=gerald carter entry in Figure 2-1. In particular, you will need to define a userPassword attribute value to further explore LDAP authentication.

Figure 2-8. person objectClass



The LDIF representation for the expanded version cn=gerald carter is:

```
dn: cn=gerald carter,ou=people,dc=plainjoe,dc=org
objectClass: person
cn: gerald carter
sn: carter
telephoneNumber: 555-1234
userPassword: {MD5}Xr4ilOzQ4PCOq3aQ0qbuaQ=  =
```

We have added an attribute named userPassword. This attribute stores a representation of the credentials necessary to authenticate a user. The prefix (in this case, {MD5}) describes how the credentials are encoded. The value in this case is simply the Base64 encoding of the MD5 hash of the word "secret."

RFC 2307 defines prefixes for several encryption algorithms. These are vendor-dependent, and you should consult your server's documentation to determine which are supported. Generating userPassword values will be covered in more detail in the context of various programming languages and APIs in later chapters. Some common encoding types are:

{CRYPT}

The password hash should be generated using the local system's crypt( ) function, which is normally included in the standard C library. The {CRYPT} prefix will be seen quite a bit in Chapter 6 when we discuss using LDAP as a replacement for NIS.

{MD5}

The password hash is the Base64 encoding of the MD5 digest of the user's password.

{SHA} (Secure Hash Algorithm)

The password hash is the Base64 encoding of the 160-bit SHA-1 hash (RFC 3174) of the user's password.

{SSHA} (Salted Secure Hash Algorithm)

This password-hashing algorithm developed by Netscape is a salted version of the previous SHA-1 mechanism. {SSHA} is the recommended scheme for securely storing password information in an LDAP directory.

2.6 Distributed Directories

At this point we have completed examining the simple directory of Figure 2-1. Since we have covered the basics, let's expand Figure 2-1 to create a distributed directory. In a distributed directory, different hosts possess different portions of the directory tree.

Figure 2-9 illustrates how the directory would look if the people ou were housed on a separate host. There are many reasons for distributing the directory tree across multiple hosts. These can include, but are not limited to:

Performance

Perhaps one section of the directory tree is heavily used. Placing this branch on a host by itself lets clients access the remaining subtrees more quickly.

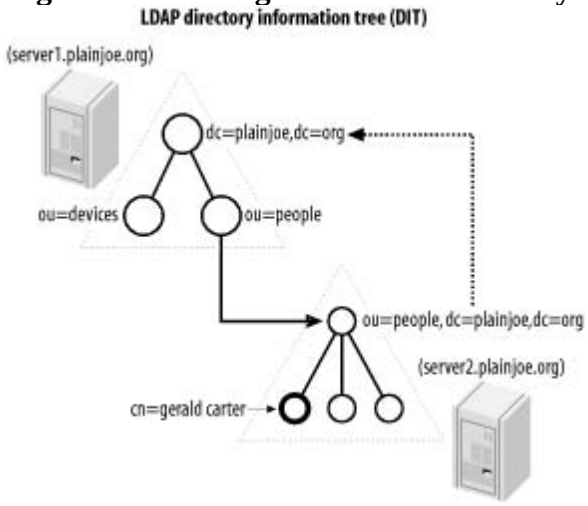
Geographic location

Are all the clients that access a particular branch of the directory in one location? If so, it would make more sense to place this section of the directory closer to the client hosts that require it. In this way, trips across a possibly slow WAN link can be avoided.

Administrative boundaries

It is sometimes easier to delegate administrative control of a directory branch by placing the branch on a server controlled by the group responsible for the information in that branch. In this way, the server operators can have full access for duties such as replication and backups without interfering with a larger, more public server.

Figure 2-9. Building a distributed directory



To divide the directory tree between the two servers in Figure 2-9, you must configure two links between the main directory server and the server that holds people ou. To do so, create the superior and subordinate knowledge reference links as shown.

A subordinate knowledge link, often called simply a *reference*, logically connects a node within a directory tree to the naming context of another server. Most often, the naming context of the second server is a continuation of the directory. In this example, the people ou in the main directory tree has no children because all queries of entries in the ou=people,dc=plainjoe,dc=org tree should be served by the second server. The entry ou=people,dc=plainjoe,dc=org on the main directory server is now a placeholder that contains the referral to the actual directory server for this entry. Figure 2-10 shows the definition for the the referral object class defined in RFC 3296.

2.7 Continuing Standardization

LDAP is continuing to evolve as a protocol. There are currently two working groups within the IETF to help shepherd this process:

- The LDAP Duplication/Replication/Update Protocols (LDUP) working group focuses on data replication and consistency in LDAP directories. More information on the group's current activities can be found at http://ietf.org/html.charters/ldup-charter.html.

- The LDAPv3 Revision (LDAPbis) working group directs its efforts toward attempting to clarify parts of the original LDAPv3 specifications. This does not include work on a Version 4 of the LDAP protocol. More information on the LDAPbis working group can be found at http://ietf.org/html.charters/ldapbis-charter.html.

While not related to standardization processes, the LDAPzone web site (http://www.ldapzone.com) does provide a nice collection of LDAP-related topics, forums, and downloads.

Chapter 3. OpenLDAP

While reading this book, you may find yourself feeling a little like a sky diver who has just jumped out of an airplane. As you approach the ground, things come more into focus. As you squint and try to make out the color of that house far below, you suddenly realize that you are plummeting closer and closer toward the very thing you are trying to observe.

Conceptual ideas need concrete implementations in order to solidify our understanding of them. A directory access protocol is of no use without an actual implementation that allows us to put the protocol to work to solve real information problems on a network. This chapter introduces OpenLDAP, a popular, open source LDAPv3-compliant server. There are a number of popular commercial products, including Sun Microsystem's SunOne directory server (formally owned by Netscape), Novell's eDirectory (formally referred to as NDS), and Microsoft's Active Directory, although this directory encompasses much more than just LDAP.

Why are we using the OpenLDAP[1] server instead of one from another vendor? OpenLDAP is attractive for several reasons:

[1] The "Open" in OpenLDAP refers to the open engineering process and community used to create OpenLDAP software.

- The OpenLDAP source code is available for download from http://www.openldap.org/ under the OpenLDAP Public License. Source code can provide a great deal of information to supplement existing (or absent) documentation.
- OpenLDAP 2 is compliant with the core LDAPv3 specifications.
- OpenLDAP is available for multiple platforms, including Linux, Solaris, Mac OS 10.2, and Windows (in its various incarnations). For more information regarding OpenLDAP on Mac OS 10.2, see http://www.padl.com//Articles/AdvancedOpenDirectoryConf.html.
- The OpenLDAP project is a continuation of the original University of Michigan LDAP server. The relationship between Michigan's LDAP server and many modern, commercial LDAP servers can be compared to the relationship between modern web browsers and the original NCSA Mosaic code base.

The examples presented in this chapter configure OpenLDAP on a Unix-based server. Therefore, they use standard Unix command-line tools such as tar, gzip, and make.

3.1 Obtaining the OpenLDAP Distribution

The OpenLDAP project does not make binary distributions of its software available. The reason for this has a lot to do with the number of dependencies it has on other packages. Many Linux vendors include precompiled versions of OpenLDAP with their distributions. Still, we'll discuss how to compile the OpenLDAP source code distribution; you'll need to build OpenLDAP to stay up to date, and studying the build process gives you a chance to learn more about the LDAP protocol.

Symas Corporation also provides some precompiled OpenLDAP packages (including requisite software components) for Solaris and HP-UX at http://www.symas.com/.

The latest version of OpenLDAP can be obtained from http://www.OpenLDAP.org/software/download/. There are two major incarnations of OpenLDAP. The older 1.2 releases are essentially enhancements or small bug fixes to the original University of Michigan code base and implement only LDAPv2. The OpenLDAP 2 branch is an LDAPv3-compliant implementation.

There are several advantages of LDAPv3 over the previous version, such as:

- The ability to refer clients to other LDAP servers for information. The LDAPv2 RFCs contained no provision for returning a referral to a client. While the University of Michigan server supported an experimental implementation of referrals, the concept was not standardized until the LDAPv3 specifications. Standardization made interoperability between servers and clients from different vendors possible, something that was missing under LDAPv2.

- The ability to publish the server's schema via LDAP operations, which makes it easier for clients to learn the server's schema before performing searches. The only way to determine the schema supported by an LDAPv2 server was to examine the server's configuration files. Publishing the server's schema as entries within the directory allows for such things as real-time updates via standard LDAP operations. (Note that LDAPv3 does not require dynamic updates.)

- Internationalization support through the use of UTF-8 characters in strings (RFC 2253) and language tags for attribute descriptions (RFC 2596).

- Improved security and flexibility for authentication credentials and data via SASL and SSL/TLS. LDAPv2 supported only simple binds or Kerberos 4 authentication.

- Support for protocol extensions as a mechanism to enhance existing operations or add new commands without requiring that a new revision of the LDAP protocol be defined.

3.2 Software Requirements

The examples presented in this book for building the client tools and server components are based on the latest OpenLDAP 2.1 release available at the current time (Version 2.1.8). As with any piece of software, version numbers and dependencies change. Make sure to consult the documentation included with future OpenLDAP releases before building your server.

Our OpenLDAP server will require several external software packages:

- Support for POSIX threads, either by the operating system or an external library.
- SSL/TLS libraries (such as the OpenSSL package, which is available from http://www.openssl.org/).
- A database manager library that supports DBM type storage facilities. The current library of choice is the Berkeley DB 4.1 package from Sleepycat Software (http://www.sleepycat.com/).
- Release 2.1 of the SASL libraries from Carnegie Mellon University (http://asg.web.cmu.edu/sasl/sasl-library.html).

3.2.1 Threads

If your server's operating system supports threads, OpenLDAP 2 can take advantage of this feature. This support works fine out of the box on most current Linux systems, Solaris, and several other platforms.

If you run into problems related to POSIX thread support, your first option is to check the OpenLDAP.org web site for installation notes specific to your platform. You may also wish to visit http://www.gnu.ai.mit.edu/software/pth/related.html for a list of known POSIX thread libraries for Unix systems. It is possible to disable thread support in the OpenLDAP server, slapd, by specifying the —disable-threads option in the OpenLDAP configure script prior to compiling. However, the replication helper daemon, slurpd, which is covered in Chapter 5, requires thread support.

3.2.2 SSL/TLS Libraries

RFC 2246 describes TLS 1.0, which resembles SSL 3.0. The StartTLS extended operation defined in RFC 2830 allows LDAP clients and servers to negotiate a TLS session at any point during a conversation (even prior to authenticating the client). To enable support for this extended operation or the LDAPS protocol, you need to obtain and install the latest version of the OpenSSL libraries. These can be downloaded from the OpenSSL Project at http://www.openssl.org/.

Building and installing the OpenSSL libraries is straightforward. Just remember that, as of release 0.9.6g, shared

3.3 Compiling OpenLDAP 2

Once the necessary software libraries have been installed and correctly configured, compiling and installing OpenLDAP becomes a matter defining the appropriate options for the configure script and executing the make command. For the sake of simplicity, all examples in this book assume that the root directory for the OpenLDAP installation is /usr/local, which is the default.

Most of the configuration options are set to reasonable defaults or will be set appropriately by the configure script itself. I've already mentioned the —disable-threads option, which you can use if you don't want thread support. You should also be aware of the —enable-wrappers option, which uses the tcp_wrappers libraries for restricting access via the standard */etc/hosts.allow* and */etc/hosts.deny*. In order to use this option, the *tcpd.h* header file and *libwrap.a* library must be installed on a local system.

For more information on tcp_wrappers, refer to the hosts_access(5) manpage or Wietse Venema's tcp_wrappers web page, which is located at ftp://ftp.porcupine.org/pub/security/index.html.

After extracting the source files using the command:

```
$ gzip -dc openldap-2.1.8.tar.gz | tar xvf -
```

go into the newly created directory and execute the ./configure script, defining any options you wish to enable or disable. For example:

```
$ cd openldap-2.1.8/
$ ./configure --enable-wrappers
```

Be sure to examine the output that follows this command to verify that the correct DBM libraries were located and any other options you defined were correctly configured. Once you are satisfied with the configuration process, building the OpenLDAP clients and servers is a four-step process:

```
$ make depend
$ make
$ make test
$ /bin/su -c "make install"
```

Here are some things to check if you have any problems:

- On systems that support it, the *ldd* tool can be used to verify that the LDAP server binary, *slapd*, is linked with the correct shared libraries. For example, if *libsasl.so* cannot be located but is installed in */usr/local/lib/*, check your system's documentation for adding directories to the library search path. Under Linux, add the directory to */etc/ld.so.conf* and rerun ldconfig -v; under Solaris (or Linux), set the LD_LIBRARY_PATH environment variable.

- Verify that DNS resolution for your host is configured correctly. In particular, reverse DNS resolution is important. Problems with DNS resolution can make it appear that the OpenLDAP server is not responding.

-

3.4 OpenLDAP Clients and Servers

The OpenLDAP package includes clients, servers, and development libraries. Table 3-1 gives an overview of the utilities that come with the package. All pathnames are relative to the installation location, which defaults to */usr/local*.

Table 3-1. Installed components included with OpenLDAP

Name	**Description**
libexec/slapd	The LDAP server.
libexec/slurpd	The LDAP replication helper.
bin/ldapadd *bin/ldapmodify* *bin/ldapdelete* *bin/ldapmodrdn*	Command-line tools for adding, modifying, and deleting entries on an LDAP server. These commands support both LDAPv2 and LDAPv3.
bin/ldapsearch *bin/ldapcompare*	Command-line utilities for searching for an LDAP directory or testing a compare operation on a specific attribute held by an entry.
bin/ldappasswd	A tool for changing the password attribute in LDAP entries. This tool is the LDAP equivalent of */ bin/passwd*.
sbin/slapadd *sbin/slapcat* *sbin/slapindex*	Tools for manipulating the local backend data store used by the *slapd* daemon.
sbin/slappasswd	A simple utility to generate password hashes suitable for use in *slapd.conf*.
*lib/liblday**	

3.5 The slapd.conf Configuration File

The *slapd.conf* file is the central source of configuration information for the OpenLDAP standalone server (*slapd*), the replication helper daemon (*slurpd*), and related tools, such as *slapcat* and *slapadd*. As a general rule, the OpenLDAP client tools such as *ldapmodify* and *ldapsearch* use *ldap.conf* (not *slapd.conf*) for default settings.

In the tradition of Unix configuration files, *slapd.conf* is an ASCII file with the following rules:

- Blank lines and lines beginning with a pound sign (#) are ignored.
- Parameters and associated values are separated by whitespace characters (space or tab).
- A line with a blank space in the first column is considered to be a continuation of the previous one. There is no need for a line continuation character such as a backslash (\).

For general needs, the *slapd.conf* file used by OpenLDAP 2 can be broken into two sections. The first section contains parameters that affect the overall behavior of the OpenLDAP servers (for example, the level of information sent to log files). The second section is composed of parameters that relate to a particular database backend used by the *slapd* daemon. It is possible to define some default settings for these in the global section of *slapd.conf*. However, any value specified in the database section will override default settings.

Here's a partial listing that shows how these two sections look:

```
 # /usr/local/etc/openldap/slapd.conf

# Global section

## Global parameters removed for brevity's sake, for now . . .

#######################################################
# Database #1 - Berkeley DB
database      bdb

## Database parameters and directives would go here.

#######################################################
# Database #2 - Berkeley DB
database      bdb

## Database parameters and directives would go here.

## And so on . . .
```

The global section starts at the beginning of the file and continues until the first database directive. We will revisit the few parameters listed here in a few moments.

The start of a database backend section is marked by the database parameter; the section continues until the beginning of the next database section or the end of the file. It is possible to define multiple databases that are served

3.6 Access Control Lists (ACLs)

The Directory ACLs provided by OpenLDAP are simple in their syntax, yet very flexible and powerful in their implementation. The basic idea is to define Who has Access to What? The most frequent forms of "Who" include:
*

Matches any connected user, including anonymous connection
self

The DN of the currently connected user, assuming he has been successfully authenticated by a previous bind request
anonymous

Nonauthenticated user connections
users

Authenticated user connections
Regular expression

Matches a DN or an SASL identity

Remember that the login name used to specify a user for authentication takes the form of a DN (e.g., dn="cn=gerald carter,ou=people,dc=plainjoe,dc=org") or an SASL identify (e.g., dn="uid=jerry,cn=gssapi,cn=auth"). The self value is used as a shortcut for the DN of the authenticated user of the current session. The examples later in this section will help clarify this concept.

The notion of an access level is a new concept. Table 3-7 summarizes the various access privileges. Higher levels possess all of the capabilities of the lower levels. For example, compare access implies auth access, and write access implies read, search, compare, and auth.

Table 3-7. Summary of access levels from most (top) to least (bottom)

Access level	**Permission granted**
write	Access to update attribute values (e.g., Change this telephoneNumber to 555-2345).
read	Access to read search results (e.g., Show me all the entries with a telephoneNumber of 555*).
search	Access to apply search filters (e.g., Are there any entries with a telephoneNumber of 555*).
compare	Access to compare attributes (e.g., Is your telephoneNumber 555-1234?).

Chapter 4. OpenLDAP: Building a Company White Pages

The previous chapter discussed how to install OpenLDAP and provided an overview of the *slapd* configuration file, *slapd.conf*. However, we have yet to launch the server, let alone add any data to the directory. Using the *slapd.conf* from Chapter 3 as a starting point, this chapter shows you how to create a company directory for storing employee contact information, including postal addresses, email addresses, and phone numbers.

4.1 A Starting Point

Here is the *slapd* configuration file developed in Chapter 3. We will change some of the entries in this listing as things progress.

```
# /usr/local/etc/openldap/slapd.conf

# Global section

## Include the minimum schema required.
include       /usr/local/etc/openldap/schema/core.schema

## Added logging parameters
loglevel      296
pidfile       /usr/local/var/slapd.pid
argsfile      /usr/local/var/slapd.args

## TLS options for slapd
TLSCipherSuite              HIGH
TLSCertificateFile          /etc/local/slapd-cert.pem
TLSCertificateKeyFile       /etc/local/slapd-key.pem

## Misc security settings
password-hash         {SSHA}

#######################################################
## Define the beginning of example database.
databasebdb

## Define the root suffix you serve.
suffix                "dc=plainjoe,dc=org"

## Define a root DN for superuser privileges.
rootdn                "cn=Manager,dc=plainjoe,dc=org"

## Define the password used with rootdn. This is the base64-encoded MD5 hash of
## "secret."
rootpw                {SSHA}2aksIaicAvwc+DhCrXUFlhgWsbBJPLxy

## Directory containing the database files
directory             /var/ldap/plainjoe.org

## Files should be created rw for the owner **only**.
mode                  0600

## Indexes to maintain
index                 objectClass          eq
index                 cn                   pres,eq

## db tuning parameters; cache 2,000 entries in memory
cachesize             2000

# Simple ACL granting read access to the world
access to *
   by * read
```

4.2 Defining the Schema

The first step in implementing a directory is determining what information to store in the directory. The naming context of your server has already been defined as:

```
dc=plainjoe,dc=org
```

Store contact information for employees in the people organzational unit:

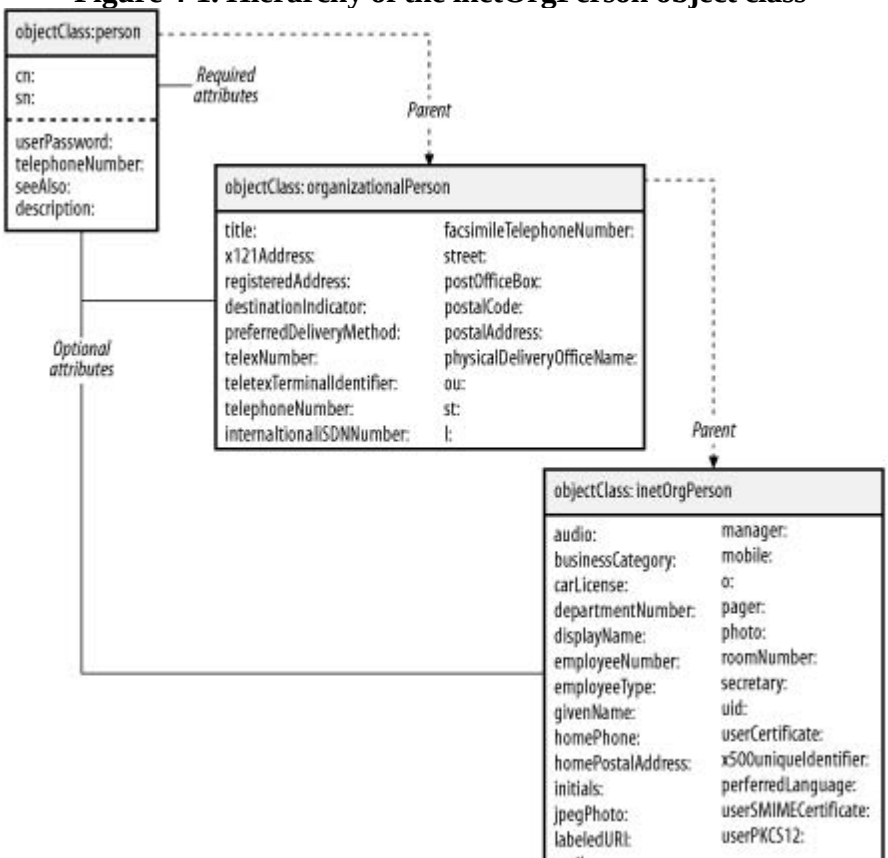
```
ou=people,dc=plainjoe,dc=org
```

There are several ways to identify the data that should be placed in an employee's entry. Information stored in an existing Human Resources database can provide a good starting point. Of course, you may not want to place all of this information in your directory. As a general rule, I prefer not to put information in a directory if that data probably won't be used. If it turns out that the data is actually necessary, you can always add it later. Eliminating unnecessary data at the start means that there's less to worry about when you start thinking about protecting the directory against unauthorized access.

An alternative to starting with an existing database is to determine which employee attributes you wish to make available and define a schema to match that list. The reverse also works: you can select a standard schema and use the attributes already defined. I prefer this approach because it makes it easy to change from one server vendor to another. Widely used, standard schemas are more likely to be supported by a wide range of vendors. Custom, in-house schemas may need to be redesigned to adapt to a new vendor (or even a new version from the same vendor).

For your directory, the inetOrgPerson schema defined in RFC 2798 is more than adequate. From Section 3.5.1 in Chapter 3, we know that this object class and associated attributes are defined in OpenLDAP's *inetorgperson.schema* file. As shown in Figure 4-1, an inetOrgPerson is a descendant of the organizationalPerson, which was itself derived from the person object class.

Figure 4-1. Hierarchy of the inetOrgPerson object class



4.3 Updating slapd.conf

Once the schema has been selected, the next step is to modify *slapd.conf* to support the selected attribute types and object classes. In order to support the inetOrgPerson object class, you must include *inetorgperson.schema*, *core.schema*, and *cosine.schema* in *slapd.conf*. The comments that begin *inetorgperson.schema* outline the dependency on the COSINE schema. Here are the modifications to the global section of *slapd.conf*:

```
# /usr/local/etc/openldap/slapd.conf

# Global section

## Include the minimum schema required.
include         /usr/local/etc/openldap/schema/core.schema

## Added to support the inetOrgPerson object.
include         /usr/local/etc/openldap/schema/cosine.schema
include         /usr/local/etc/openldap/schema/inetorgperson.schema

## Added logging parameters
 . . .
```

The database section is currently in working condition, so only a few changes are needed. To better support searches for employees, you should modify the set of indexes to include a more complete list of attributes. In addition to creating an index for the cn attribute, you'll also index the surname (sn) and email address (mail) attributes. In addition to the equality (*eq*) index, you'll add a substring (*sub*) index to support searches such as "All employees whose last names begin with C." Finally, you will add an equality index for the departmentNumber attribute so that users can search for employees within a given department. This index would not be necessary if the directory were laid out as shown in Figure 4-2 because the same effect could be achieved by beginning the search at the department ou. Here are the changes to the database section:

```
## Indexes to maintain
index         objectClass         eq
index         cn,sn,mail          eq,sub
index         departmentNumber    eq
 . . .
```

At this point, it's a good idea to verify that the location specified by the directory parameter exists and has the proper permissions. In our example, that directory is */var/ldap/plainjoe.org*. If this directory does not exist, the following two commands ensure that the filesystem is ready to store data:

```
root# mkdir -p /var/ldap/plainjoe.org
root# chmod 700 /var/ldap/plainjoe.org
```

4.4 Starting slapd

Once the final tweaks have been added to the configuration file, the next step is to start the *slapd* daemon by executing the following command as root:

```
root# /usr/local/libexec/slapd
```

Use the ps command to verify that *slapd* is running. On a Linux system, the output should appear similar to:

```
$ ps -ef | grep slapd
root   8235     1  0 12:37 ?   00:00:00 /usr/local/libexec/slapd
root   8241  8235  0 12:37 ?   00:00:00 /usr/local/libexec/slapd
root   8242  8241  0 12:37 ?   00:00:00 /usr/local/libexec/slapd
```

On Linux and IRIX, multiple threads of a process will show up as individual entries in the output from ps. On Solaris, *slapd* will be displayed as a single process.

Stopping the OpenLDAP server requires that the daemon have a chance to flush modified directory data to disk. The best way to do this is to send the parent slapd process an INT signal, as shown here (the *pidfile* location was defined in the server's configuration file):

```
root# kill -INT 'cat /var/run/slapd.pid'
```

Shutting down *slapd* by more drastic means, such as kill -9, can result in data corruption and should be avoided at all costs.

In the absence of any command-line options, *slapd*'s behavior is governed by compile-time defaults or options defined in the *slapd.conf* file. At times, it is necessary to override some of these settings via the command line. Table 4-1 lists the available *slapd* options.

Table 4-1. Command-line options for the slapd server

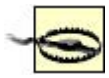
Option	**Description**
-d integer	Specifies the log level to use for logging information. This option causes *slapd* to log all information to standard output on the controlling terminal; it can be very helpful for quick server debugging sessions. The integer value specified should be a combination of the logging levels associated with the loglevel parameter in *slapd.conf*.
-f filename	Uses a configuration file other than the compile-time default (*slapd.conf*).
-h URI_list	Specifies a space-separated list of LDAP URIs that the *slapd* daemon should serve. The most common URIs are ldap:/// (LDAP on port 389; the default), ldaps:/// (LDAP over SSL on port 636), and ldapi:/// (LDAP over IPC).
	Specifies the local user of the syslog facility. The default

4.5 Adding the Initial Directory Entries

A directory without data isn't of much use. There are two ways to add information to your directory; which method to use depends on the directory's state. First, *slapadd* and the other *slap** commands were presented in Chapter 3 as database maintenance tools. They allow an administrator to import entries directly into the database files and export the entire directory as an LDIF file. They work directly with the database, and don't interact with slapd at all. Second, the OpenLDAP distribution includes a number of tools, such as *ldapmodify*, that can update a live directory using the LDAPv3 network operations. These tools access the directory through the server.

What are the advantages and disadvantages of these approaches? The offline tools can be much faster; furthermore, there are circumstances when you can't start the server without first adding data (for example, when restoring the directory's contents from a backup). The disadvantage of the offline tools, of course, is that they must be run locally on the server.

In contrast to the offline tools, the LDAP client utilities are more flexible and allow a directory administrator greater control by forcing user authentication and by using access control lists on directory entries. A good rule of thumb is that the *slap** tools are used for getting your LDAP server online, and the *ldap** tools are for day-to-day administration of the directory.



OpenLDAP 2.1 removed the restriction that *slapd* must not be running before any of the *slap** tools can be used. However, OpenLDAP 2.0 caches data in memory, so using the slap* tools while the directory is running can present an inconsistent view of the directory at best, and corrupt data at worst.

The tools for offline manipulation of directory information are *slapadd*, *slapcat*, *slapindex*, and *slappasswd* (covered in Chapter 3). The *slapadd* tool determines which files and indexes to update based on the *slapd.conf* file. Because it is possible for a given configuration file to define more than one database partition, *slapadd* provides options for specifying a database partition by either the directory suffix (-b suffix) or the numbered occurrence (-n integer) in *slapd.conf*. Referring to a particular database using a numbered instance is confusing and error-prone. It is far more intuitive to refer to the same database by using the directory suffix. Note that the -b and -n options are mutually exclusive. A summary of the various *slapadd* command-line options is provided in Table 4-2.

Table 4-2. Summary of slapadd command-line arguments

Option	**Description**
-c	Continues processing input in the event of errors.
-b suffix-n integer	Specify which database in the configuration file to use by the directory's suffix (-b) or by its location (-n) in the *slapd.conf* file (the first database listed is numbered 0). These options are mutually exclusive.
	Specifies which debugging information to log. See the

4.6 Graphical Editors

Working with command-line tools and LDIF files is constructive, but certainly not convenient. There are a number of graphical editors and browsers for LDAP that make it easier to see what you're doing. I won't discuss any of these in detail, but I'll give you some pointers to some tools that are worth looking at:
GQ (http://biot.com/gq/)

GQ is a GTK+-based LDAPv3 client capable of browsing the subSchema entry on LDAPv3 servers. It is distributed under the GNU GPL and includes features such as:

- Support for browsing or searching LDAP servers
- Support for editing and deleting directory entries
- Support for creating template entries based on existing ones
- Support for exporting subtrees or an entire directory to an LDIF file
- Support for multiple server profiles
- SASL authentication

Java LDAP Browser/Editor (http://www.iit.edu/~gawojar/ldap/)

This is an editor written in Java using the JNDI class libraries. It supports:

- LDAPv2 and v3 servers, including SSL connections
- Editing attribute values
- Searching for specific entries
- Exporting and importing data using LDIF files

Chapter 5. Replication, Referrals, Searching, and SASL Explained

The previous chapters have prepared the foundation for understanding and building an LDAP-based directory server. This chapter presents some of the more advanced features provided by LDAP and shows how to use these features in your directory service. As such, this chapter ties up a lot of loose ends that have been left hanging by the previous discussions.

5.1 More Than One Copy Is "a Good Thing"

We begin by exploring directory replication. This feature hasn't been standardized yet; our example focuses on the OpenLDAP project. The concepts and principles that I will present are applicable to all LDAP directories, but the examples themselves are specific to OpenLDAP.

Because LDAP replication is vendor-specific at the moment, it is not possible to replicate data from one vendor's server to another vendor's server. It is possible to achieve cross-vendor replication by using tricks such as parsing a change log, but these tricks are themselves vendor-dependent.

The LDAP Duplication/Replication/Update Protocols (LDUP) Working Group of the IETF attempted to define a standardized replication protocol that would allow for interoperability between all LDAPv3-compliant servers. However, there appears to be more demand for an LDAP client update protocol (LCUP) that would allow clients to synchronize a local cache of a directory as well as be notified of updates. Details of the group's progress can be found at http://www.ietf.org/html.charters/ldup-charter.html.

A frequently asked question is: "When should I install a replica for all or part of my directory?" The answer depends heavily on your particular environment. Here are a few symptoms that indicate the need for directory replicas:

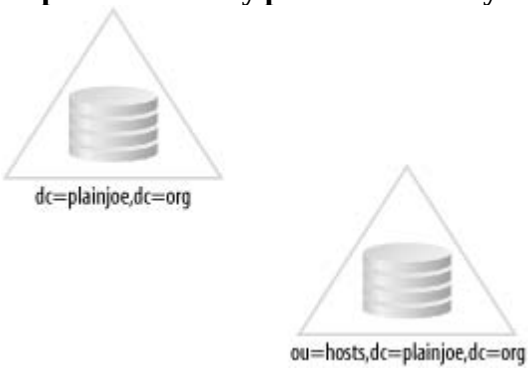
- If one application makes heavy use of the directory and slows down the server's response to other client applications, you may want to consider installing a replica and dedicating the second server solely to the application that is causing the congestion.
- If the directory server does not have enough CPU power to handle the number of requests it is receiving, installing a replica can improve response time. You may also wish to install several read-only replicas and use some means of load balancing, such as round-robin DNS or a virtual server software package. Before taking this route, make sure that the limiting factor is CPU and not other finite resources such as memory or disk I/O.
- If a group of clients is located on the other side of a slow network link, installing a local replica (i.e., a replica that is close to the clients making the request) will decrease traffic over the link and improve response time for the remote clients.
- If the directory server cannot be taken offline for backups, consider implementing a read-only replica to provide service while the master server is taken down for backups or normal maintenance.
- If your directory is a critical part of the services provided by your network, using replicas can help provide failover and redundancy.

5.2 Distributing the Directory

The scenarios presented thus far have all assumed that the entire directory consists of a single partition on one server. In the real world, this may not always suffice. There are many reasons (which I touched on in Chapter 2) for splitting a directory into two or more partitions, which may reside on multiple servers.

Let's assume that, according to Figure 5-2, the top level of your directory server (dc=plainjoe,dc=org) is maintained by one department, and the server containing host information (ou=hosts,dc=plainjoe,dc=org) is managed by another. How can these two directories be combined into one logical DIT?

Figure 5-2. Two separate directory partitions held by different servers



The definition for the ou=hosts partition held by the second server is very similar to the database section we have been using so far. The main changes are to the suffix served by the backend (ou=hosts,dc=plainjoe,dc=org) and the directory in which the BerkeleyDB files are stored (*/var/ldap/hosts/*). The rootdn (cn=Manager,ou=hosts,dc=plainjoe,dc=org) must also be updated due to the requirement that it must exist within the partition's naming context.

```
#######################################################
## Partition on second server holding ou=hosts
database        bdb

## Define the root suffix you serve.
suffix          "ou=hosts,dc=plainjoe,dc=org"

## Define a root DN for superuser privileges.
rootdn          "cn=Manager,ou=hosts,dc=plainjoe,dc=org"

## Define the password used with rootdn. This is the Base64-encoded MD5 hash of
## "secret."
rootpw          {SSHA}2aksIaicAvwc+DhCrXUFlhgWsbBJPLxy

## Directory containing the database files
directory       /var/ldap/hosts

## Files should be created rw for the owner **only**.
mode            0600

## Indexes to maintain
index           objectClass     eq
index           cn              pres,eq

## db tuning parameters; cache 2,000 entries in memory
cachesize       2000

# Simple ACL granting read access to the world
access to *
     by * read
```

5.3 Advanced Searching Options

Chapter 4 presented LDAP searches as a means of verifying the correctness of your directory. That's obviously a very limited use of the search capability: a directory isn't much use if you can't search it. Given our limited goals in the previous chapter, we didn't do justice to the topic of search filters. It's now time to take a more thorough look at the topic of filters.[1]

[1] For the full details of representing LDAP searches using strings, read RFC 2254.

In its commonly used form, an LDAP search filter has the following syntax:

```
( attribute filterOperator value )
```

The *attribute* is the actual name of the attribute type. The *filterOperator* is one of:

=

For equality matches

~=

For approximate matches

<=

For less than comparisons

>=

For greater than comparisons

If you deal only with string comparisons, you may only need the equality operator.

The *value* portion can be either an absolute value, such as carter or 555-1234, or a pattern using the asterisk (*) character as a wildcard. Here are some wildcard searches:

(cn=*carter)

Finds all entries whose cn attribute ends in "carter" (not just those with a last name of Carter)

(telephoneNumber=555*)

Finds all telephone numbers beginning with 555

You can combine single filters like these using the following Boolean operators:

&

Logical AND

|

Logical OR

!

Logical NOT

5.4 Determining a Server's Capabilities

Chapter 2 alluded to two new LDAPv3 features: the subschemaSubentry and the rootDSE objects. Both of these objects allow clients to find out information about a previously unknown directory server.

The rootDSE object contains information about features such as the server naming context, implemented SASL mechanisms, and supported LDAP extensions and controls. LDAPv3 requires that the rootDSE has an empty DN. To list the rootDSE, perform a base-level search using a DN of "". OpenLDAP will provide only values held by the rootDSE if the search requests that operational attributes be returned, so the + character is appended to the search request.

```
$ ldapsearch -x -s base -b "" "(objectclass=*)" +

dn:
structuralObjectClass: OpenLDAProotDSE
namingContexts: dc=plainjoe,dc=org
supportedControl: 2.16.840.1.113730.3.4.2
supportedControl: 1.3.6.1.4.1.4203.1.10.2
supportedControl: 1.2.826.0.1.334810.2.3
supportedExtension: 1.3.6.1.4.1.4203.1.11.3
supportedExtension: 1.3.6.1.4.1.4203.1.11.1
supportedExtension: 1.3.6.1.4.1.1466.20037
supportedFeatures: 1.3.6.1.4.1.4203.1.5.1
supportedFeatures: 1.3.6.1.4.1.4203.1.5.2
supportedFeatures: 1.3.6.1.4.1.4203.1.5.3
supportedFeatures: 1.3.6.1.4.1.4203.1.5.4
supportedFeatures: 1.3.6.1.4.1.4203.1.5.5
supportedLDAPVersion: 3
supportedSASLMechanisms: GSSAPI
supportedSASLMechanisms: DIGEST-MD5
supportedSASLMechanisms: CRAM-MD5
subschemaSubentry: cn=Subschema
```

This list can change over time and will vary from server to server. Our example shows us that this server supports:

- StartTLS (OID 1.3.6.1.4.1.1466.20037) and two other extended operations
- ManageDsaIT (OID 2.16.840.1.113730.3.4.2) and two other LDAP controls
- LDAPv3 operations only
- The GSSAPI, DIGEST-MD5, and CRAM-MD5 SASL mechanisms
- A single naming context of "dc=plainjoe,dc=org"

There may be additional attributes and values, depending on the LDAP server.

5.5 Creating Custom Schema Files for slapd

There are times when the standard schema files distributed with your LDAP server don't meet the needs of your application. Creating a custom schema file for OpenLDAP is a simple process:

- Assign a unique OID for all new attribute types and object classes.
- Create the schema file and include it in *slapd.conf*.

It's also possible to create alternate schema syntaxes and matching rules, but implementing them is beyond the scope of this book; typically, they require implementing a plug-in for the directory server or modifying the server's source code. For more information on this process, you should consult the OpenLDAP source code or your vendor's documentation for other directory servers.

Chapter 2 described how to obtain a private enterprise number from IANA (see the form at http://www.iana.org/cgi-bin/enterprise.pl and RFC 3383). When creating new attributes or object classes, it is a good idea to use an OID that is guaranteed to be unique, whether or not the schema will ever be used outside of your organization. The best way to guarantee that the OID is unique is to obtain a private enterprise number and place all your definitions under that number.

For example, suppose that an LDAP client application requires a new object class based on person. This new object class should contain all of the attributes possessed by the person object, with the addition of the userPassword and mail attributes.

In order to create this new object, I have allocated the OID arc of 1.3.6.1.4.1.7165.1.1.1 for the new object classes:

```
iso (1)
  org (3)
    dod (6)
      internet (1)
        private (4)
          enterprise (1)
            SAMBA.org (7165)
              plainjoe.org (1)
                O'Reilly LDAP Book(1)
```

The private enterprise number 7165 has been issued by IANA for use by the Samba developers, the 7165.1 arc has been allocated to the plainjoe.org domain, and 7165.1.1 has been set aside for this book; I can't touch the numbers above 7165.1 in the tree, but I have complete freedom to assign numbers below it as I see fit. I've chosen to allocate 7165.1.1.1 to ldap object classes that I create and 7165.1.1.2 for new attributes. I could put my new objects directly under plainjoe.org, but that might cause problems if I want to create other kinds of objects (for example, private SNMP MIBs):

```
SAMBA.org (7165)
  plainjoe.org (1)
    O'Reilly LDAP Book(1)
    |-- objectclasses  (1)
    |-- attributeTypes (2)
```

Let's call the new object plainjoePerson. Add the following definition to a custom schema file named

5.6 SASL and OpenLDAP

The final section of this chapter explores how to replace the simple authentication used in your current directory server with SASL mechanisms. You will be using the GSSAPI mechanism for Kerberos 5 authentication (RFCs 1510, 2743, and 2478). The examples assume that a Kerberos realm named PLAINJOE.ORG has already been established and that a service principal named *ldapadmin* has been created. If you are unclear on the details of Kerberos 5, a good place to start would be Kerberos: A Network Authentication System, by Brian Tung (Addison-Wesley), or The Moron's Guide to Kerberos, located at http://www.isi.edu/gost/brian/security/kerberos.html.

So far, the rootdn and rootpw values used in *slapd.conf* have appeared similar to:

```
rootdn          "cn=Manager,dc=plainjoe,dc=org"
rootpw          {SSHA}2aksIaicAvwc+DhCrXUFlhgWsbBJPLxy
```

In OpenLDAP 2.1, an SASL ID can be converted to a distinguished name and used for authentication or authorization wherever a normal DN would be appropriate. This includes operations such as defining the updatedn used for replication or the binddn used by a client in a search request. There's one important exception to this rule: don't use an SASL ID as the DN of an entry in the directory. To summarize from Chapter 3, an SASL ID converted to a DN appears as:

```
uid=name[,realm=realm],cn=mechanism,cn=auth
```

To illustrate how to use SASL as the authentication mechanism, we'll replace the rootdn in our master server's *slapd.conf* with the Kerberos 5 principal *ldapadmin*. Following the conversion algorithm just discussed, the new rootdn in *slapd.conf* will be:

```
## New SASL-based rootdn
rootdn      "uid=ldapadmin,cn=gssapi,cn=auth"
```

The rootpw entry can be deleted because authentication for the new rootdn will be done using the SASL GSSAPI mechanism. The OpenLDAP server must possess a valid keytab file containing the key for decrypting tickets transmitted with client requests.[2] Moreover, our tests will assume that the server is configured to use the default realm of *PLAINJOE.ORG*.

[2] More information on generating keytab files can be found on the *kadmin(8)* manpage.

Once the configuration change has been made, restart *slapd*. You can then verify that the change has been made correctly by using the *ldapadd* command to add an entry; the rootdn is currently the only DN allowed to write to the directory.

To run this test, create a file with an LDIF entry; we'll use the following LDIF entry, stored in */tmp/test.ldif*:

```
## Test user to verify that the new rootdn is OK.
dn: cn=test user,ou=people,dc=plainjoe,dc=org
cn: test user
sn: test
objectclass: person
```

To add this entry to the directory, invoke *ldapadd* with some additional arguments:

```
$ kinit ldapadmin@PLAINJOE.ORG
Password for ldapadmin@PLAINJOE.ORG: password
$ klist
Ticket cache: FILE:/tmp/krb5cc_780
Default principal: ldapadmin@PLAINJOE.ORG
```

Part II: Application Integration

Chapter 6. Replacing NIS

One of LDAP's chief advantages is its ability to consolidate multiple directory services into one. This chapter examines the pros and cons of using LDAP as a replacement for Sun's Network Information Service (NIS). NIS is used primarily by Unix clients to centralize management of user information and passwords, hostnames and IP addresses, automount maps (files that control the mounting of remote file systems), and other administrative information. NIS clients for other operating systems, such as Windows NT 4.0, exist, though they aren't particularly common.[1]

[1] NIS was superseded by NIS+, which was never widely adopted. Describing how to replace NIS+ is beyond the scope of this book.

While the focus of this chapter is using an LDAP directory as a replacement for NIS domains, many other tools are used to distribute management information on Unix systems; for example, many sites use *rsync(1)* to push administrative files, such as /etc/passwd, to client machines. While this chapter assumes that you are replacing NIS with an LDAP directory, adapting these techniques I present to other schemes for sharing the data in /etc/passwd, /etc/hosts, and other key files should be straightforward:

- Get the information you want to share into the directory.
- Get your clients to use the directory.
- Disable your old information-sharing mechanism.

There are two fundamental strategies for replacing NIS with an LDAP directory. The first solution, illustrated in Figure 6-1, involves setting up an NIS/LDAP gateway: i.e., an NIS server that accepts NIS queries, but answers the queries by retrieving information from an LDAP directory. This strategy doesn't require any client modifications, and therefore works with all NIS clients; it may be the easiest means of transitioning to a new LDAP-based information service. Sun Microsystems Directory Server 4.x supports this approach. Sadly, newer releases (5.x) of Sun's directory services product do not. An alternative solution is the NIS/LDAP gateway provided by a company named PADL Software (http://www.padl.com/). This gateway product is available for servers running Solaris, Linux, FreeBSD, or AIX, and will be discussed later in this chapter.

Figure 6-1. NIS/LDAP gateway

The second solution involves making a complete transition to LDAP. If you are willing to disable NIS lookups on all of your clients and install the necessary LDAP libraries and modules, you may prefer this approach. Clients access information directly from an LDAP directory, eliminating the gateway. Many modern operating systems support

6.1 More About NIS

Before discussing these strategies for replacing NIS with LDAP, it's worth understanding something about the beast we're trying to replace.[3] NIS is most commonly used to distribute system password and account maps (i.e., */etc/passwd* and */etc/shadow*) to client machines. It's also used to distribute the information from many other system files, such as */etc/hosts*, */etc/services*, */etc/group*, and */etc/networks*. It can also distribute a number of files that control the automatic mounting of remote file systems; and with the appropriate wizardry in sed and awk or Perl, it can be coerced into distributing almost any kind of data that can be represented in a text file.

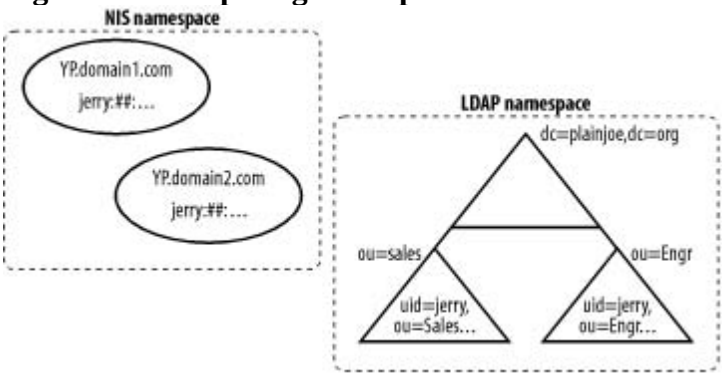
[3] This discussion is necessarily very brief. If you need more information about NIS, see Managing NFS and NIS, by Hal Stern, Mike Eisler, and Ricardo Labiaga (O'Reilly).

In the NIS world, the master copy of any shared data resides on a master server, which distributes the data to slave servers. Clients, which are organized into NIS domains (not to be confused with DNS domains), can then access this information from any NIS server, master or slave, that services their domain. The NIS master acts as a directory system agent (DSA) that provides information to clients, which use this information to perform tasks such as authenticating users (i.e., the passwd map) and locating other hosts on the network (i.e., the hosts map).

The NIS information model is also characterized by a flat namespace. To use the *passwd.byname* map as an example (this map represents the */etc/passwd* file, indexed by username), there can be only one login name of *jerry*. To work around this deficiency, NIS groups client machines into NIS domains, each with its own set of maps (i.e., its own set of virtual administrative files). So, two users with the login name *jerry* can coexist if they can be placed in different domains; for all practical purposes, different NIS domains are different directories (even though they may be served by the same server).

In contrast, LDAP allows you to create a hierarchical namespace to manage these users. Let's assume that we'll use the RDN of a node as its login name. LDAP can then maintain multiple users with the same login name if we make sure that each user belongs to a different parent node (see Figure 6-2).

Figure 6-2. Comparing namespaces in NIS and LDAP



One basic rule of system administration is that users should not notice any loss of service when you implement changes. A user does not need to be aware of where their account information is stored. It makes little difference to them, as long as they can access necessary network services. If a change results in a downgrade of service for users (no matter how big of a win for the system administrators), you'll almost certainly be forced to rip it out and go back to the old system; eventually, you'll get tired of answering all the help desk calls. Fortunately, the flexibility of the PAM and NSS interfaces can do a lot to insulate users from a change in information location.

6.2 Schemas for Information Services

RFC 2307, "An Approach for Using LDAP as a Network Information Service," which has recently been updated in an Internet-Draft by the LDAPbis working group, defines the attribute types and object classes needed to use an LDAP directory as a replacement for NIS. Despite its experimental status, several vendors such as Sun, Apple, HP, SGI, OpenLDAP, and PADL Software have developed products that support this schema.

RFC 2307 relates directly to information stored in standard NIS maps and how these maps should be viewed by directory-enabled client applications. The list of attribute types and object classes is lengthy; for a complete description of all that is available, refer to the RFC. I will use portions of the RFC 2307 schema in examples later in this chapter. Before trying to implement these examples or experimenting with this schema on your own, consult your directory server's documentation to find out the server's level of support for RFC 2307 and the exact syntax you should use for working with RFC 2307 objects.

The first example shows how to migrate all user accounts and groups into your OpenLDAP server. While there is nothing out of the ordinary about the configuration parameters with which you'll implement this solution, here's a complete listing of the revised slapd.conf; note that two new schema files are included, nis.schema (the RFC 2307 schema) and cosine.schema (which defines items required by nis.schema):

```
## slapd.conf for implementing an LDAP-based Network Information Service

## Standard OpenLDAP basic attribute types and object classes
include      /usr/local/etc/openldap/schema/core.schema

## cosine.schema is a prerequesite of nis.schema.
include      /usr/local/etc/openldap/schema/cosine.schema

## rfc2307 attribute types and object classes
include      /usr/local/etc/openldap/schema/nis.schema

## Misc. configure options
pidfile      /var/run/slapd.pid
argsfile     /usr/run/slapd.args
loglevel     256

## SSL configure options
TLSCipherSuite          3DES:RC4:EXPORT40
TLSCertificateFile      /usr/local/etc/openldap/slapd-cert.pem
TLSCertificateKeyFile   /usr/local/etc/openldap/slapd-private-key.pem

#########################################################
## Define the beginning of example database.
database     bdb
suffix       "dc=plainjoe,dc=org"

## Define a root DN for superuser privileges.
rootdn          "cn=Manager,dc=plainjoe,dc=org"
rootpw          {SSHA}2aksIaicAvwc+DhCrXUFlhgWsbBJPLxy

## Directory containing the database files
directory     /var/ldap/plainjoe.org
mode          0600

## Create the necessary indexes.
index     objectClass     eq

## These indexes are included to support calls such as getpwuid(  ), getpwnam(  ), and
## getgrgid(  ).
index     cn,uid        eq
index     uidNumber     eq
```

6.3 Information Migration

While some organizations may have the resources (such as undergraduate work study students) to re-enter the data held in the NIS maps to the LDAP store, luckily, there are other means available. In addition to the PAM and NSS LDAP reference modules available at PADL Software's web site, you'll also find a set of Perl scripts designed to convert the various */etc* system files (e.g., */etc/passwd* and */etc/hosts*) into LDIF format. Once you've converted the system files to LDIF, you can import them into your LDAP store either online using the *ldapadd(1)* command or by using an offline database creation utility such as the OpenLDAP *slapadd(8c)* tool. These LDAP migration scripts can be found at http://www.padl.com/OSS/MigrationTools.html.

After unpacking the migration scripts, you must customize the *migrate_common.ph* script to fit your network settings. Within this Perl script is a variable named $DEFAULT_BASE, which is used to define the base suffix under which the organizational units that will serve as containers for migrated information will be created.

The scripts accept input and output filenames as command-line parameters. If no output filename is present, the scripts write the converted entries to standard output. For example, the following command converts */etc/passwd* into an LDIF file:

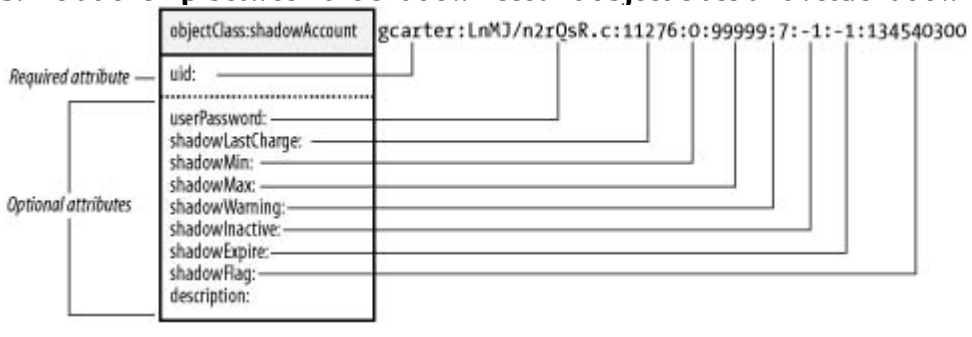
```
root# migrate_passwd.pl /etc/passwd /tmp/passwd.ldif
```

Here's what a typical entry from */etc/passwd* looks like after it has been translated:

```
dn: uid=gcarter,ou=people,dc=plainjoe,dc=org
uid: gcarter
cn: Gerald Carter
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
userPassword: {crypt}LnMJ/n2rQsR.c
shadowLastChange: 11108
shadowMax: 99999
shadowWarning: 7
shadowFlag: 134539460
loginShell: /bin/bash
uidNumber: 780
gidNumber: 100
homeDirectory: /home/gcarter
gecos: Gerald Carter
```

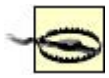
All the required fields (cn, uid, uidNumber, gidNumber, and homeDirectory) defined in the RFC schema for a posixAccount are present. There are also a number of shadow fields (shadowLastChange, etc.; see the shadowAccount object in Figure 6-5), which hold values related to password aging. These values are taken automatically from the */etc/shadow* file. If your system doesn't use shadow passwords, the shadowAccount object class values may not be present.

Figure 6-5. Relationship between the shadowAccount object class and /etc/shadow file entry



6.4 The pam_ldap Module

Pluggable Authentication Modules (PAM) are implemented as shared libraries that distance applications from the details of account data storage, mechanisms used to authenticate users, and service authorization processes. PADL Software has developed a pam_ldap module, supported on FreeBSD, HP-UX, Linux, Mac OS 10.2, and Solaris, as part of a reference implementation for RFC 2307. This module allows you to take advantage of LDAP in PAM-aware applications and operating systems. You can download the pam_ldap source code from http://www.padl.com/OSS/pam_ldap.html. Most Linux distributions include PADL's pam_ldap and nss_ldap modules with the operating system. You should remove these packages first if you plan to build the latest version from source.



The pam_ldap and nss_ldap libraries included with Solaris as part of the operating system installation are Sun's own creation and should not be confused with the modules discussed in this chapter.

Once you have obtained and extracted the pam_ldap source code, building the module is a familiar process:

```
$ ./configure
$ make
$ /bin/su -c "make install"
```

PADL's PAM and NSS libraries can make use of the Netscape LDAP SDK and the original University of Michigan LDAP SDK, in addition to the OpenLDAP client libraries. The configure script attempts to determine which of these packages is installed on the local system. If you need to inform the configure script which LDAP client libraries you have installed and where, use the following configure options:

```
--with-ldap-lib=type   select ldap library [auto|netscape3|
                       netscape4|umich|openldap]
--with-ldap-dir=DIR    base directory of ldap SDK
```

6.4.1 Configuring /etc/ldap.conf

The pam_ldap module (and as we will see shortly, PADL's nss_ldap module) stores its configuration settings in a text file. This file is named *ldap.conf* by default and is normally stored in the */etc* directory. The configuration parameters you can put in this file are summarized in Table 6-1, Table 6-2, and Table 6-3; the list is fairly small and self-explanatory. We will begin customizing this file by exploring how a client locates the LDAP server and authenticates itself.

Table 6-1. ldap.conf parameters shared by pam_ldap and nss_ldap

Parameter	**Description**
host	The IP address (or hostname) of the LDAP server. The value must be resolvable without LDAP support. If a host is not specified, the nss_ldap library will attempt to locate an LDAP server by querying DNS for an SRV record for _ldap._tcp.<domain>. The current version of the pam_ldap module (v157) will not perform this auto-lookup, but support is planned for a future release. Also refer to the uri parameter.

6.5 The nss_ldap Module

The Name Service Switch (NSS) is similar to PAM except that it only provides a mechanism for information retrieval. PADL Software's nss_ldap module can be obtained from http://www.padl.com/OSS/nss_ldap.html. The current implementation can be used on AIX, HP-UX, Linux, and Solaris. Although the pam_ldap module supports FreeBSD and Mac OS 10.2, the nss_ldap library does not. This means that you will not be able to apply the complete solution outlined in this chapter to those platforms.

Compiling PADL's nss_ldap module is almost the same as compiling pam_ldap. The same options are available to the configure script (for explicitly defining the LDAP libraries to be used and their locations). The one additional compile-time setting that you will use is *-enable-rfc2307bis*. This change optimizes the search parameters for each nsswitch.conf database by using the *nss_base_** parmeters. Otherwise, nss_ldap would query for entries by performing a subtree search beginning at the *base* (from */etc/ldap.conf*). The familiar three-step:

```
$ ./configure  --enable-rfc2307bis
$ make
$ /bin/su -c "make install"
```

installs an appropriately named version of the nss_ldap library in */lib*. For example, the resulting file would be */lib/libnss_ldap.so* on a Linux host and */lib/nss_ldap.so* on a Solaris box. Since the examples in this chapter are based on Linux systems, whenever there is a need to refer to the actual nss_ldap library file, I will use the *libnss_ldap.so* filename.

The nss_ldap module uses the same */etc/ldap.conf* configuration file as PADL's pam_ldap module. The configuration parameters for this module are summarized in Table 6-3. While both pam_ldap and nss_ldap read */etc/ldap.conf* for configuration settings, the parameters prefixed by pam_ do not affect the behavior of nss_ldap.

The */etc/ldap.conf* file must be readable by any process that performs any of the various getXbyY( ) function calls such as getpwnam*(*jerry*)* or getgrgid(0). For example, if you have specified a host to which all LDAP queries should be directed, but the user's process is unable to obtain that parameter setting because it cannot read ldap.conf, you will begin to notice DNS SRV queries for _ldap._tcp.*domain* as the nss_ldap library attempts to locate an LDAP server. However, if you make the ldap.conf file world-readable, think twice about putting a binddn and bindpw in the file.

To configure a service to use the nss_ldap module, add the keyword ldap to the appropriate lines in your */etc/nsswitch.conf* file. PADL's NSS module currently supports the following databases:
passwd group hosts services networks protocols rpc ethers netgroups

The following databases are currently unsupported:
netmasks bootparams publickey automount

Mount point lookups using LDAP queries are supported directly by some automount agents, such as Sun's automounter (included with current Solaris releases) and Linux's autofs. This will be covered later in the chapter.

Here's an excerpt from an nsswitch.conf file. It specifies that the system should consult the local password, shadow password, and group files before querying the directory server.

```
## Define the order of lookups for users and groups.
passwd:     files ldap
shadow:     files ldap
group:      files ldap
```

6.6 OpenSSH, PAM, and NSS

Once the pam_ldap and nss_ldap shared libraries have been installed and */etc/ldap.conf* has been configured, you can configure individual services to use the new PAM module. We'll start with the SSH daemon, *sshd*. Here's how to set up OpenSSH (http://www.openssh.com/) on a Linux system, which uses a separate PAM configuration file per service. (Note that other systems may use a single PAM file for all services; for example, Solaris uses /etc/pam.conf.) Make sure that PAM is enabled when you compile the *sshd* daemon; otherwise, you will be wasting your time.

The following */etc/pam.d/sshd* configuration file defines the pam_ldap library to be used for authentication (auth) and account management (account). The account management library checks for password aging according to the attribute types defined for the shadowAccount object class and verifies any host-based access rules (covered in the next section). The session module type is ignored by the pam_ldap library. While user password changes are supported by the pam_ldap library, these are not relevent to this example.

```
 ## /etc/pam.d/sshd
## PAM configuration file for OpenSSH server
auth       required     /lib/security/pam_nologin.so
auth       sufficient   /lib/security/pam_ldap.so
auth       required     /lib/security/pam_unix.so shadow nullok use_first_pass

account    sufficient   /lib/security/pam_ldap.so
account    required     /lib/security/pam_unix.so

password   required     /lib/security/pam_cracklib.so
password   required     /lib/security/pam_unix.so nullok use_authtok  shadow

session    required     /lib/security/pam_unix.so
session    optional     /lib/security/pam_console.so
```

The use of the sufficient control flag for the auth and account service types indicates that authentication by this module alone is enough to return success to the invoking application. The use_first_pass argument is necessary so that the user is not prompted for an additional password if authentication falls through to the *pam_unix.so* library.

You will have to create a similar configuration file for every other service for which you want to control access.

While configuring sshd to use PAM for authentication requires some configuration, nothing needs to be done to make *sshd* use the nss_ldap library. The retrieval of information from the various databases listed in /etc/nsswitch.conf is handled by the system's standard C library; once you've set up nsswitch.conf, you're done. The client application only needs to call the basic get . . . ( ) function, such as getpwnam( ), to obtain the available information.

6.7 Authorization Through PAM

Once a user has been authenticated, the account management features of the pam_ldap module provide two means of restricting access to a host, independent of any other PAM modules you may have specified in the configuration file (e.g., the pam_nologin module). Which method you choose depends on whether you wish to bind a host to a group of users or bind a user to a group of hosts.

6.7.1 One Host and a Group of Users

The first authorization method, in which you specify a group of users who are allowed to use a particular host, ties into other information you have already migrated into the directory. The host entry for a machine (generated from */etc/hosts* by the PADL migration scripts) can be extended to include a list of DNs for users (member) that are authorized to log on using pam_ldap. The following LDIF example shows how you can use the extensibleObject class to associate a group of users with a host entry:

```
dn: cn=pogo,ou=hosts,dc=plainjoe,dc=org
objectClass: ipHost
objectClass: device
objectClass: extensibleObject
ipHostNumber: 192.168.1.75
cn: pogo.plainjoe.org
cn: pogo
member: uid=gcarter,ou=people,dc=plainjoe,dc=org
member: uid=kristi,ou=people,dc=plainjoe,dc=org
member: uid=deryck,ou=people,dc=plainjoe,dc=org
```

In order to configure pam_ldap to honor this group membership, the following two lines must be added to */etc/ldap.conf*:

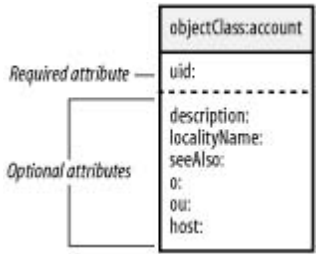
```
## Define the DN of the entry to contain the groupOfUniqueNames.
pam_groupdn              cn=pogo,ou=hosts,dc=plainjoe,dc=org

## Define the attribute type that should be used in the attempt to match the user's
## DN.
pam_member_attribute       member
```

For OpenSSH, this configuration means that only those users whose DN is listed as one of the values for the member attribute will be allowed ssh access to your host.

6.7.2 One User and a Group of Hosts

You can also specify the machines that any given user is allowed to access. To implement this control mechanism, the structural account object class listed in the *cosine.schema* file must be present in the list of object classes for an entry. This is done for you by the PADL migration scripts. Figure 6-6 shows that the account object class requires only one attribute. This attribute, uid, is already required by the posixAccount object class and is therefore guaranteed to be present.

Figure 6-6. Schema for the account object class



6.8 Netgroups

Netgroups have become a daily staple for NIS administrators. They allow machines and/or users to be collected together for various administrative tasks such as grouping machines together for use in the tcp_wrappers files */etc/hosts.allow* and */etc/hosts.deny*. In this next example, you restrict access via ssh only to members of the *sysadmin* netgroup:

```
# /etc/hosts.deny
sshd: ALL
 . . .
# /etc/hosts.allow
sshd: @sysadmin
```

Netgroups can be composed solely of individual hosts:

```
sysadmin  (garion.plainjoe.org,-,-)(silk.plainjoe.org,-,-)
```

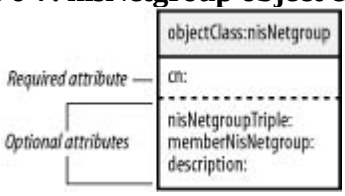
or other netgroups:

```
all_sysadmin    sysadmin secure_clients
```

or of any combination of the two.

RFC 2307 describes the structural nisNetgroup object class (Figure 6-7), which can be used to represent netgroups as directory entries. The cn attribute holds the name of the netgroup, the nisNetgroupTriple attribute stores the (host, user, NIS-domain) entries, and the memberNisNetgroup attribute stores the names of any nested netgroups.

Figure 6-7. nisNetgroup object classes



Before adding any netgroup entries to the directory, you must create the container ou. By convention, I will use the ou=netgroup organizational unit for storing netgroups in this example:

```
dn: ou=netgroup,dc=plainjoe,dc=org
objectclass: organizationalUnit
ou: netgroup
```

After passing through PADL's *migrate_netgroup.pl* tool, the *sysadmin* netgroup will be represented by this LDIF entry:

```
$ ./migrate_netgroup.pl /etc/netgroup
dn: cn=sysadmin,ou=netgroup,dc=plainjoe,dc=org
objectClass: nisNetgroup
objectClass: top
cn: sysadmin
nisNetgroupTriple: (garion.plainjoe.org,-,-)
nisNetgroupTriple: (silk.plainjoe.org,-,-)
```

The *all_sysadmin* netgroup contains the *sysadmin* and the *secure_clients* netgroups, so it will use the memberNisNetgroup attribute:

```
dn: cn=all_sysadmin,ou=netgroup,dc=plainjoe,dc=org
objectClass: nisNetgroup
objectClass: top
cn: all_hosts
memberNisNetgroup: sysadmin
```

6.9 Security

Up to this point, we haven't discussed security. You've put a lot of sensitive information into your directory, which is now controlling whether users can log into machines on your network. And you could certainly put a lot more information into the directory: telephone numbers, human resources information, etc. Some of this information might be genuinely useful to the public at large; some of it may be highly confidential. But you don't yet know how to keep users from accessing information they shouldn't have access to. In order to have any confidence in a solution, we must examine how certain security issues are addressed by both the PAM and NSS modules.

First, it is important to understand what level of security is desired and exactly what information is being protected. Are you concerned only with protecting passwords? What about usernames as well? From the perspective of system administration, the most important information to protect is related to user and group accounts. Few sysadmins worry about someone being able to snoop a hosts file as it is copied across the network from one machine to another. However, everyone should be concerned about using a clear-text protocol, such as FTP, to transfer /etc/passwd and /etc/shadow from one machine to another.

To protect user passwords, we must look at how the PAM module binds to the directory. pam_ldap always uses a simple bind to authenticate a user against an LDAP server. You should avoid sending account credentials across the network in a form that is readable by anyone viewing traffic.

LDAPv3 provides two mechanisms that can be used to protect passwords. One is to use SASL to support more secure methods of authentication such as Kerberos 5 or DIGEST-MD5. However, while this mechanism protects passwords, it doesn't necessarily protect information other than the user's password. It is not supported by pam_ldap at this time. The second solution is to negotiate a secure transport layer that will protect the information used in the LDAP bind request as well as all other information sent to and from the directory server.

Security must be implemented by both the server and the client. It makes no difference if one party is willing to communicate securely but the other is not. Recall the StartTLS-extended command added in RFC 2830. This command allows the client to request a secure transport layer prior to binding to an LDAP server. Both the pam_ldap and nss_ldap modules support using the StartTLS command to negotiate transport layer security. In addition, these modules also support the LDAPS (tcp/636) protocol, which is an older method for accessing an LDAP server securely.

The following ldap.conf directive instructs the PADL LDAP modules to issue a StartTLS command prior to binding to the server:

```
 ## Use the StartTLS command to negotiate an encrypted transport layer. A value of
## on defines the use of LDAPS when connecting to the directory server.
ssl     start_tls
```

Once you have configured the client, use a tool such as Ethereal or tcpdump to view the network traffic; it's a good idea to verify that things are working as expected.[6] After the initial LDAP Extended Request (i.e., StartTLS), you should see no clear-text traffic between the client and server. It is easy to spot an LDAP simple bind. The following is a bind request using the DN "uid=gcarter,ou=people,dc=plainjoe,dc=org" and the password testing:

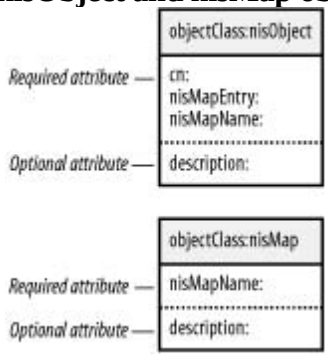
[6] More information on Ethereal can be found at http://www.ethereal.com/. News regarding tcpdump can be found at http://www.tcpdump.org/.

```
 00 30 f1 11 98 da 00 00 f4 d8 6c 0d 08 00 45 00   .0........l...E.
00 71 b9 a2 40 00 40 06 fd 21 c0 a8 01 4a c0 a8   .q..@.@..!...J..
01 28 a3 2f 01 85 26 8e 13 41 30 62 21 3a 80 18   .(./..&..A0b!:..
```

6.10 Automount Maps

In order to use the automount information stored in your directory, you must shift your focus to the automount daemon itself, specifically Linux's kernel-based autofs. As it currently stands, autofs (v3.1.7 and the 4.0 preview releases) supports the undocumented automount and automountMap object classes. However, Red Hat has updated the package in its distribution (autofs-3.1.7-28) to look up mount points based on the nisObject and nisMap classes described in RFC 2307 (and included in *nis.schema*). The LDAPbis workgroup's revisions to RFC 2307 will include new schema items for storing automount information, but for the moment, nisObject and nisMap have the largest support base from Red Hat, Sun, and PADL. Figure 6-8 shows the required and optional attributes for these two new object classes.

Figure 6-8. nisObject and nisMap object classes



Red Hat's automount patches can be obtained from either http://people.redhat.com/nalin/autofs/ or in the latest autofs SRPM at ftp://ftp.redhat.com/pub/redhat/linux/rawhide/SRPMS/SRPMS/.

PADL's migration tools include a script (*migrate_automount.pl*) for converting an automount map to LDIF. Here, you will convert a single automount point in */opt* to a directory entry. You can see from the */etc/auto.opt* excerpt that the LDIF entry contains all of the information needed for mounting */opt/src*. This time, PADL's script does create the top-level container (nisMapName=auto.opt) for you:

```
 $ grep src /etc/auto.opt
src    -rw,hard,intr    queso.plainjoe.org:/export/u1/src

$ ./migrate_automount.pl /etc/auto.opt /tmp/auto.opt.ldif

$ cat /tmp/auto.opt.ldif
dn: nisMapName=auto.opt,dc=plainjoe,dc=org
objectClass: top
objectClass: nisMap
nisMapName: auto.opt

dn: cn=src,nisMapName=auto.opt,dc=plainjoe,dc=org
objectClass: nisObject
cn: src
nisMapEntry: -rw,hard,intr queso.plainjoe.org:/export/u1/src
nisMapName: auto.opt
```

After adding the new automount entries to the directory using *ldapadd*, the autofs server must be informed of *auto.opt*'s map location, the LDAP server's hostname, and the search base. The following line in */etc/auto.master* instructs the autofs package to look up mounts for */opt* on the host *ldap1* beneath ou=auto.opt,dc=plainjoe,dc=org:

```
 ## Look up mounts for /opt in the LDAP directory.
   /opt  ldap:ldap1:nisMapName=auto.opt,dc=plainjoe,dc=org  --timeout 300
```

6.11 PADL's NIS/LDAP Gateway

If configuring all your Unix clients to use PAM and installing the various NSS modules is a little more work than your IT shop can bear at the moment, you may prefer the NIS/LDAP gateway solution mentioned at the beginning of this chapter (refer to Figure 6-1 for an illustration). This section examines PADL Software's ypldapd daemon as a migration path from NIS- to directory-based information storage. The following excerpt from the *ypldapd(8)* manpage describes ypldapd's position within a network:

YPLDAP(8)

ypldapd emulates the equivalent process ypserv by providing an RPC call-compatible interface. Rather than consulting `map' files as ypserv does, however, ypldapd draws its data from LDAP databases.

In theory, ypldapd allows an NIS domain to be replaced with a directory-based solution without any client machines being aware of the change. Even non-Unix NIS clients, such as the Windows NT NISgina DLL, will function correctly. As far as NIS clients are concerned, nothing has changed: they still get their data using the NIS protocol from an NIS server. Where the server gets its data from is another matter.

The ypldapd package is available in binary form for Solaris, Linux, FreeBSD, and AIX, and can be downloaded with a 30-day evaluation license. PADL's web site provides instructions for obtaining a temporary license via an email request. The user's guide is also available online in either Postscript or MS Word format (http://www.padl.com/Products/NISLDAPGateway.html).

Configuring ypldapd is fairly easy. Because it supports the RFC 2307 information service schema, you can use the PADL migration tools described earlier in this chapter to populate the directory with host and user information. PADL includes a copy of its migration tools with the ypldapd distribution. However, you may want to download the latest version separately.

PADL provides installation scripts for ypldapd that can be executed after unpacking the tar archive in */opt/ypldapd*. Before beginning the installation, you should have or know:

- A license key for *ypldapd*
- The hostname of the LDAP server to query
- The base DN used for searches
- The NIS domain name of the *ypldapd* server

Chapter 7. Email and LDAP

One of the most important applications of a directory is storing email addresses and other contact information. Although many ad hoc solutions to this problem have been implemented over the years, LDAP provides a natural online publishing service for this type of data. This chapter explores the ins and outs of integrating email clients (MUAs) and mail servers (MTAs) with an LDAP directory. It covers the configuration details of some of the more popular email clients, including Mozilla Mail, Pine, Microsoft Outlook, and Eudora. We'll also discuss the schema required to support these clients and the types of LDAP searches to expect when the application attempts to locate a user in the directory.

On the server side, we'll discuss three popular email servers—Sendmail, Postfix, and Exim—all of which can use a directory. We will cover the level of LDAP support within each MTA, the schema needed to support this integration, and the configuration process for integrating an LDAP directory into a production email environment. This discussion assumes that you are familiar with basic MTA administration and the interaction between SMTP servers.

7.1 Representing Users

The server you will build combines the white pages server you created in Chapter 4 and the server for administrative databases you created in Chapter 6 as a replacement for NIS. You already have a head start on integrating user account information because both servers used the ou=people container for storing user account information. With only a few modifications to your directory, the posixAccount and inetOrgPerson object classes can be used to store a single user entry for both authentication and contact information.

Here's an entry for "Kristi Carter," which is similar to those presented in Chapter 4:

```
dn: cn=Kristi W. Carter,ou=people,dc=plainjoe,dc=org
objectClass: inetOrgPerson
cn: Kristi W. Carter
sn: Carter
mail: kcarter@plainjoe.org
labeledURI: http://www.plainjoe.org/~kristi
roomNumber: 102 Ramsey Hall
telephoneNumber: 222-555-2356
```

In Chapter 6, this same user might have been presented as:

```
dn: uid=kristi,ou=people,dc=plainjoe,dc=org
uid: kristi
cn: Kristi Carter
objectClass: account
objectClass: posixAccount
userPassword: {crypt}LnMJ/n2rQsR.c
loginShell: /bin/bash
uidNumber: 781
gidNumber: 100
homeDirectory: /home/kristi
gecos: Kristi Carter
```

Looking at both examples side by side, some differences can be noted. The first is that the RDN used for each entry is different. It doesn't really matter whether you choose cn=Kristi W. Carter or uid=kristi. Since Unix accounts must already possess a unique login name, the uid attribute is a good choice to prevent name conflicts in ou=people.

The second issue is more serious and shows why the initial directory design should not be rushed. Both the account and inetOrgPerson object classes are structural object classes. Remember that an entry cannot have more than one structural object class and that once an entry is created, its structural class cannot be changed. Some LDAP servers may allow you to reassign an entry's object classes at will, but do not rely on this behavior.

To solve this dilemma, initially create each entry with the inetOrgPerson class and then extend it using the posixAccount auxiliary class. The means that the account entry will have to filtered from the output of PADL's migration scripts—a simple task using *grep*:

```
$ ./migrate_passwd.pl /etc/passwd | \
  grep -iv "objectclass: account" > passwd.ldif
```

The combined user entry now appears as:

```
dn: uid=kristi,ou=people,dc=plainjoe,dc=org
objectClass: inetOrgPerson
objectClass: posixAccount
cn: Kristi Carter
cn: Kristi W. Carter
sn: Carter
mail: kcarter@plainjoe.org
labeledURI: http://www.plainjoe.org/~kristi
```

7.2 Email Clients and LDAP

When planning a strategy for supporting an application with a directory, you always start by examining the application and determining what schema has the ability to support it. Using a standard schema is vastly preferable to building your own. Of course, with email you don't have the ability to specify what client users will use: at your site, many different clients are probably in use, and you won't make friends by asking users to change. In this section, we'll look at four clients, all of which are in common use: Mozilla Mail, Pine from the University of Washington, Qualcomm's Eudora, and Microsoft's Outlook Express. Fortunately, the inetOrgPerson schema supports all of the information items we are concerned with using in this section.

The following parameters are common to all clients:

- The LDAP server is *ldap.plainjoe.org*.
- The base search suffix is ou=people,dc=plainjoe,dc=org.

Beyond the basic LDAP search parameters and supporting schema, it is imperative to know what version of LDAP the clients will use. Table 7-1 reveals that 3 out of the 4 mail clients listed use LDAPv2 to bind to the directory server. This means that you must explicitly add support for these connections as OpenLDAP 2.1 rejects LDAPv2 binds in default configurations. Add the following line to the global section of *slapd.conf*:

```
## Allow LDAPv2 binds from clients needed by several mail client packages.
allow      bind_v2
```

then restart the OpenLDAP server to make it recognize the change.

Table 7-1. LDAP versions used by various mail clients

Mail client	**LDAPv2 bind**	**LDAPv3 bind**
Mozilla Mail	✓	
Pine 4	✓	
Eudora	✓	
Outlook Express		✓

7.2.1 Mozilla Mail

In 1998, Netscape Communications announced that it would give away the source code to the next generation of

7.3 Mail Transfer Agents (MTAs)

The remainder of this chapter discusses LDAP support within several popular MTAs. You can skim this material if you want an overview of various mail servers, or you can focus on the details regarding your specific MTA and skip the others. In either case, I assume that you have some familiarity with the Simple Mail Transport Protocol (SMTP) and mail servers in general.

Before we begin, Table 7-2 provides a summary of the LDAP versions used by the mail servers presented in this section. The same rule for enabling LDAPv2 binds described in the beginning of this chapter still holds true for two out of the three mail servers listed.

Table 7-2. LDAP versions used by various mail servers

Mail transfer agent	LDAPv2 bind	LDAPv3 bind
Sendmail	✓	
Postfix		✓
Exim	✓	

7.3.1 Sendmail

Sendmail is the default MTA on most current versions of Unix. A number of alternatives have appeared in the past few years (such as Postfix, Qmail, and Exim), but if you work with Unix or Linux systems, chances are you'll deal with Sendmail. Sendmail introduced support for retrieving information from an LDAP directory in Version 8.9. However, this support didn't really stabilize until later versions (this discussion focuses on Version 8.12). It by no means attempts to give comprehensive coverage of Sendmail. For information on the details of configuring and running a Sendmail server, refer to Sendmail, by Bryan Costales and Eric Allman (O'Reilly).

Sendmail's LDAP integration falls into four categories: support for retrieving mail aliases from a directory, support for accessing generic Sendmail maps using LDAP queries, expansion of Sendmail classes at startup using information obtained from a directory, and support for retrieving specific mail-routing information from an LDAP directory. We will return to the specifics of these features in later sections.

In order to support any of these functions, Sendmail must be compiled with the LDAPMAP option enabled. Here's how to modify site.config.m4 to compile LDAP against the client libraries installed by OpenLDAP 2:

```
dnl  . . . /devtools/Site/site.config.m4
dnl Enable LDAP features in Sendmail during compilation
dnl
APPENDDEF (`confMAPDEF , `-DLDAPMAP )dnl
APPENDDEF(`confLIBS , `-lldap -llber )dnl
dnl
dnl The following two entries are needed so
dnl that make can find the the ldap header files
```

Chapter 8. Standard Unix Services and LDAP

In Chapter 6, we examined the possibilities of integrating an LDAP directory into basic authentication services by using the PAM and NSS modules. In Chapter 7, we integrated LDAP into the network mail infrastructure in both clients and servers. This chapter takes LDAP integration a step further by exploring how other standard Unix services can make use of our directory. The applications we will explore are Apache, FTP (ProFTPD), Samba, RADIUS (FreeRadius), DNS (BIND 9), and printing (LPRng and LPD). It is impossible to cover all the services a network may provide, but by showing a few concise, real-world solutions to common problems, I hope to give you tools and ideas that you can apply to any network applications you encounter in the future.

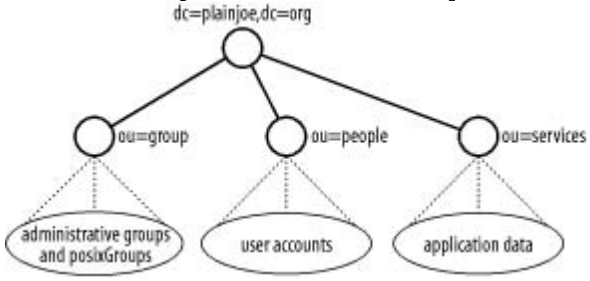
The applications discussed in this chapter will communicate directly with the LDAP directory. Servers that do not possess native LDAP support can use the PAM and NSS solutions presented in Chapter 6.

8.1 The Directory Namespace

Although we will eventually need to modify it, we will start by using the namespace developed in Chapter 6 and Chapter 7. Our directory root is dc=plainjoe,dc=org, and user-related information is stored beneath the people organizational unit directly below the root, as shown in Figure 8-1. The group organizational unit contains any posixGroup entries as well as any administrative group (groupOfNames) objects used in access control rules for the directory.[1] We will need to add additional organizational units, which we will do later in this chapter.

[1] Examples of using administrative groups in ACLs and the groupofNames object can be found in Appendix E.

Figure 8-1. Initial namespace for the directory used in this chapter

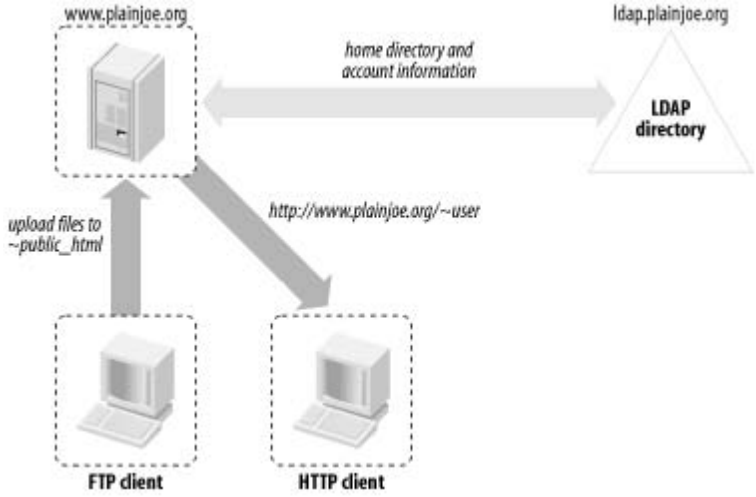


8.2 An FTP/HTTP Combination

The first set of services that we will explore is the combination of the ProFTPD server (http://www.proftpd.org/) and Apache (http://www.apache.org/). In this scenario, we would like to build a new web server and allow users to publish web content using an FTP client. All user and group accounts already exist in the LDAP directory, but just to make things interesting, assume that your theoretical web server platform cannot make use of either PAM or NSS to access any of this information.

The solution we would like to deploy is illustrated in Figure 8-2. Users should be able to put files into *~<username>/public_html* on the web server using FTP. ProFTPD must authenticate user connections using information stored in the LDAP directory. These files should then be accessible via a web browser at http://www.plainjoe.org/*~<username>*. Because the server is not using an nss_ldap library, Apache must obtain the home directory path for users directly from the LDAP server.

Figure 8-2. Interaction between ProFTPD, Apache, and the LDAP directory on www.plainjoe.org



Two add-ins, both developed by John Morrissey (http://www.horde.net/~jwm), will help you implement your new web server. We will begin by looking at ProFTPD's LDAP features.

8.2.1 ProFTPD

Morrissey's LDAP authentication module (mod_ldap) is included with current releases of the ProFTPD server.[2]

[2] ProFTPD's mod_ldap module should not be confused with any of Apache's modules of the same name (http://modules.apache.org/).

Our focus will be on the ProFTPD Release v1.2.7. Building mod_ldap is controlled by an option to the configure script (—with-modules=mod_ldap). After extracting the source code, the following commands build the binaries and tools to include support for mod_ldap:

```
$ ./configure --with-modules=mod_ldap
$ make
$ /bin/su -c "make install"
```

You can specify multiple modules as arguments to the —with-modules option by separating each module name in the list with a colon (e.g., —with-modules=mod_ldap:mod_linuxprivs). For more information on the various modules supported by the ProFTPD package, refer to the documentation included with the software or the online versions at http://www.proftpd.org/docs/.

8.3 User Authentication with Samba

This book has concentrated on Unix services, with only a few exceptions; email applications often cross platform boundaries, as do requirements for file and printer sharing. The Samba project (http://www.samba.org/) has become a staple for administrators seeking to integrate Unix file and print servers with Windows clients. Samba is a suite of programs that implement the server portion of the SMB (Server Message Block) protocol, later renamed CIFS (Common Internet File System).

Samba includes several client programs and administrative tools in addition to its server components. Adequate coverage of Samba is well beyond the scope of this book. For more information about Samba, see Sams Teach Yourself Samba in 24 Hours, Second Edition, by Gerald Carter (Sams Publishing), or Using Samba, Second Edition, by Jay Ts, Robert Eckstein, and David Collier-Brown (O'Reilly).

To support the challenge/response authentication methods used by Microsoft clients, Samba requires a list of hashed passwords separate from the normal Unix account information stored in */etc/passwd* (or in the posixAccount object class). This collection of LanManager and Windows NT password hashes is normally stored in a file named *smbpasswd(5)*; the format of each entry is:

```
username:uid:LM_HASH:NT_HASH:account flags:timestamp
```

Samba's *smbpasswd* file has several disadvantages for sites with many users:

- Lookups are performed sequentially. When servicing a domain logon request from a Windows NT/2000/XP client, there are a minimum of two lookups. These lookups can be a performance bottleneck.
- Attempts at using a single *smbpasswd* file for multiple standalone servers requires the administrator to use external tools, such as a combination of *rsync(1)* and *ssh(1)* or *scp(1)*, to replicate the file. This solution also requires that the set of Unix users and groups be synchronized between the servers, perhaps using the methods outlined in Chapter 6.
- The format of the *smbpasswd* file limits the number of attributes that can be maintained for each user. When Samba is acting as a Windows Primary Domain Controller, there are many additional fields, such as the location of a user's roving profile, that should be maintained on an individual basis.

8.3.1 Configuring Samba

All of these deficiencies can be addressed by moving the information from a local, flat file into sambaAccount objects in an LDAP directory. The LDAP support in Samba 2.2.7a must be enabled at compile time using the —with-ldapsam configure script option.[3]

[3] The LDAP support in Samba 2.2 has no relationship to the LDAP support in a Windows 2000 domain or in Windows 2000 Active Directory servers.

8.4 FreeRadius

The FreeRadius server project (http://www.freeradius.org/) is the implementation of the Remote Authentication Dial-In User Service (RADIUS) protocol used by many corporations and Internet service providers to authenticate users connecting from remote locations. Complete coverage of FreeRadius or RADIUS servers goes beyond the scope this chapter. RFC 2865 explains the details of the protocol. For a more practical look at RADIUS, you should refer to the FreeRadius web site as well as RADIUS, by Jonathon Hassel (O'Reilly).

The FreeRadius server daemon, *radiusd*, can use an LDAP directory in two different ways. First, it can use LDAP as a data store for RADIUS attribute values. RADIUS attributes are defined by the RADIUS protocol and should not be confused with LDAP attributes.[4] The only similarity between the two types of attributes is that both have names and are used to store values. The FreeRadius administrator defines the mapping between RADIUS attributes and the LDAP attributes used to represent them. We'll look at the configuration details after we have compiled a working RADIUS server. The second option is to use the directory as an authentication service by binding to the LDAP server on behalf of a user. In this way, *radiusd* can determine whether to accept or reject incoming connection requests.

[4] A list of RADIUS attributes linked with the corresponding RFCs can be found at http://www.freeradius.org/rfc/attributes.html.

In the 0.8 release, the rlm_ldap module used by *radiusd* to access a directory is included in a default installation. No additional flags are required to enable LDAP support at compile time. Running the basic configure && make && /bin/su -c "make install" is enough to achieve a working *radiusd* in most environments.

Without getting too bogged down in the specifics of the FreeRadius configuration file, *radiusd.conf*, it is worth explaining the general layout. Configuration options can be described as either existing within the scope of a section bounded by { }s or global. Global parameters define information such as the location of directories necessary to the general operation of radiusd or the number of threads that the main server should spawn. Scoped parameters can be subdivided into module settings and component implementations.

FreeRadius modules are shared libraries defined by the project's RLM interface. The modules block in *radiusd.conf* contains parameters specific to each library. The RLM interface describes several different components that a module can implement. The two components of interest to us are authorize and authenticate.

The authorization component is used by *radiusd* to look up information about a user account. The authorize section can contain several different module names. Each module is queried in order for an entry matching the login name of the user in question until a record is located or all modules have reported failure. Part of the authorization component's responsibility is to describe the authentication method used to validate this account. The authenticate section defines possible authentication mechanisms. The method actually used for a specific request is determined by the information returned by the authorize section.

Here is the working configuration file for a basic server to authenticate connections against the list of local accounts:

```
 ## radiusd.conf: FreeRADIUS server configuration file

##
## Global parameters: directory/logfile locations, etc.
##
prefix = /opt/radius
```

8.5 Resolving Hosts

Now let's turn our attention to data describing hosts on a network. One of the most fundamental services provided in any TCP/IP network is the resolution of machine names to network addresses. The most widespread mechanism for looking up IP addresses is the Domain Name System (DNS). Again, coverage of DNS is beyond the scope of this book; for more information, see DNS and BIND, Fourth Edition, by Cricket Liu and Paul Albitz (O'Reilly).

Chapter 1 already made it clear that LDAP is not a replacement for a specialized directory service such as DNS. However, you can use LDAP effectively as a backend storage system for DNS zone files. Stig Venaas has written such a patch for Bind 9 using its new simplified database interface (SDB). The latest release of the patch for BIND 9.1 (or later) and the necessary schema file for OpenLDAP 2 can be obtained from http://www.venaas.no/ldap/bind-sdb/. For performance reasons, I recommend that you obtain the latest patch, rather than using the one included in the *contrib/* subdirectory of the latest BIND 9 release.

Venaas has included a brief list of the steps necessary for integrating LDAP-sdb support in Bind 9. Here are the instructions contained in the *INSTALL* file of the ldap-sdb archive:

1. Copy the *ldap.c* source file to the *bin/named/* subdirectory of the BIND 9 source tree.

2. Copy the *ldap.h* header file to the *bin/named/include/* subdirectory of the BIND 9 source tree.

3. Edit *bin/named/Makefile.in* and add the following lines:

```
DDRIVER_OBJS = ldapdb.@O@
DDRVIVER_SRCS = ldapdb.c
DDRIVER_LIBS = -lldap -llber
```

   The Makefile variables may already exist; if this is the case, simply append the references to the LDAP files to the existing definitions. You may also need to add the path to the LDAP include files and libraries to the *DDRIVER_INCLUDES* and *DDRIVER_LIBS* respectively.

4. Edit *bin/named/main.c* and add the line #include <ldapdb.h> below #include "xxdb.h"; add the line ldapdb_init( ); below xxdb_init( ); and finally, add ldapdb_clear( ); below xxdb_clear( );.

After making these changes, you're ready to build the LDAP-enabled *named* binary by executing ./configure && make && /bin/su -c "make install". It is a good idea to ensure that the new server is working with an existing set of zone files before continuing. Here is a zone file for the plainjoe.org domain; it contains four hosts, *localhost*, *dns1*, *ldap*, and *ahab*:

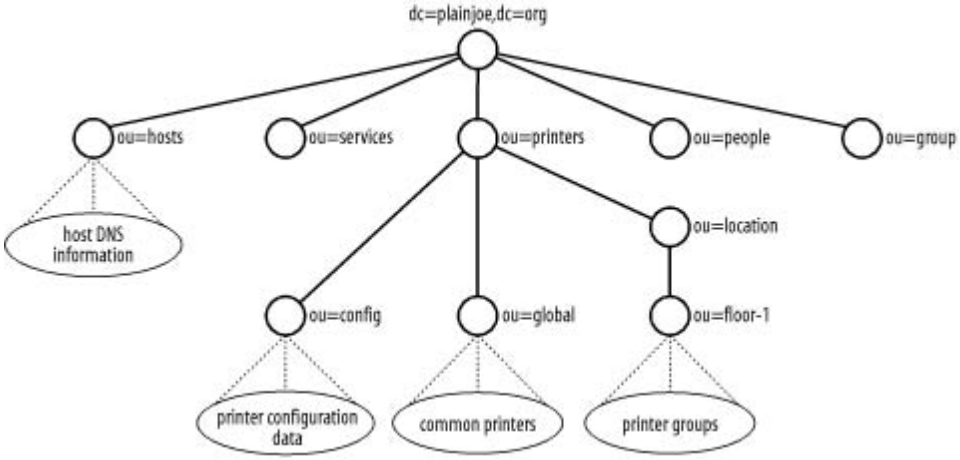
```
 plainjoe.org. IN SOA dns1.plainjoe.org. root.dns.plainjoe.org. (
                      3         ; Serial
                      10800     ; Refresh after 3 hours
                      3600      ; Retry after 1 hour
                      604800    ; expire after 1 week
                      86400 )   ; minimum TTL of 1 day
; Name Servers
plainjoe.org.               IN     NS     dns1.plainjoe.org.
```

8.6 Central Printer Management

Now that you've moved your DNS zone data into an LDAP directory, you have the leisure to ask, "What's the big deal?" DNS already has highly effective mechanisms for distributing and replicating zone data; it's not like user account data, which needs to be kept consistent on every machine. So have you accomplished anything, aside from being able to point to a directory server that's serving the zone data to your DNS servers? Clearly, you need to be able to justify the effort you've spent, and to do so, you need to find another application that can make use of the same data.

Network printers are devices that are associated with entries in DNS and possess additional attributes used to support a non-DNS application (i.e., printing). Our next step is to design a directory-based solution for managing printer configuration information that simplifies the process of adding, deploying, and retiring printers. A printer should be accessible to its clients as soon as it has been added to the directory. The namespace shown in Figure 8-8 was designed with this philosophy in mind. All printer configuration information is stored below the ou=printers organizational unit. The immediate three children, config, global, and location, are used to group printers and maintain configuration parameters.

Figure 8-8. LDAP namespace for directory-based storage of printer configuration data



The config organizational unit sits at the root of the actual configuration tree. Each printer has an entry containing information such as the printer's name and maximum print job size. For network printers, the entry also contains DNS information, such as IP address and hostname. The ou=config,ou=printers,dc=plainjoe,dc=org entry acts as the base suffix for the *lp.plainjoe.org* DNS zone used by your BIND 9 server. This means that if an administrator removes a printer's entry from the config organizational unit, it is immediately removed by DNS as well. Devices that are physically connected to a host acting as the spooler are not considered network printers for the purposes of this discussion.

Printers listed beneath the ou=global entry should be available to all clients on the network. The entries here contain only a printer's name; the actual configuration data can be accessed by querying for the attributes of cn=*printername* ,ou=config, ou=printers,dc=plainjoe,dc=org. The ou=location tree has a similar function to the global tree. The location organizational unit is a holder for another group of organizational units, one of which is shown in Figure 8-8. Each organizational unit at this level represents a group of printers. Each client on the network can list one or more group names; the clients are then allowed to access the printers in the groups that they have listed.

The major difficulty in dealing with printers is deciding on an acceptable schema for representing printer capabilities and data. Currently, there is no standardized printer schema. The closest we have to a standard is the Internet-Draft *draft-fleming-ldap-printer-schema-XX.txt*. We only need to implement a subset of the schema in this document (see Figure 8-9).[5] The *printer.schema* file also includes a modified version of the schema presented in Network

Chapter 9. LDAP Interoperability

What is a chapter on interoperability doing in a book on LDAP? After all, I've presented LDAP throughout this book as a standard protocol, and standards are supposed to minimize, if not eliminate, interoperability problems. One of the major selling points of LDAP is its potential for consolidating vendor-specific or application-specific directories. We've seen many examples of this: using LDAP as a replacement for NIS, as a backend data store for DNS, and as a replacement for many ad hoc databases used in email management.

Still, while LDAP minimizes interoperability problems, "minimize" is definitely the key word. The core features of LDAP are standardized, but things such as schemas are not. There are many common object classes and attributes that can be extended by a vendor. Not only can schemas be extended, the protocol can be extended as well by creating additional operations using extensions and controls, and not all vendors support the same ones.

For each service that can be consolidated into an LDAP directory, there must be a corresponding client-side application that can access the old information in the new directory. That's not always an easy order to fill; we've already seen some clever workarounds to help older applications access an LDAP directory, such as using the pam_ldap library presented in Chapter 6 to enable non-LDAP-aware applications to authenticate users in the directory. Furthermore, sooner or later you will encounter an LDAP-enabled application that requires the directory service to implement a specific schema or extended operation.

The goal of this chapter is to discuss several technologies that you can use to solve problems of this sort. Every directory integration project is unique. I will show how to solve a number of common directory integration problems—and although the problems I discuss are typical enough, they're only a small fraction of the problems you're likely to face. The most effective way to prepare yourself to solve the problems posed by your environment is to examine the tools, concepts, and architectures that can be combined into a solution to meet the needs of your users.

9.1 Interoperability or Integration?

The terms interoperability and integration each have a different place within our coverage of LDAP. For our purposes, directory integration means enabling client applications to access data in an LDAP directory, a topic that has been covered extensively in previous chapters. Interoperability should address communication between LDAP servers themselves. The distinction between integration and interoperability begins to blur when one LDAP server becomes the client of another LDAP server.

Whenever you start thinking about interoperability or integration, your first step should be to ask what level of interoperability or integration your application requires. There are a number of solutions that provide interoperability or integration in various forms. Knowing what your application requires will make it much easier to decide which solution is appropriate. Table 9-1 lists some common approaches to interoperability and integration issues.

Table 9-1. Common directory interoperability solutions

Problem	**Possible solution**	**Example**
"What can I do if my application doesn't speak LDAP?"	Gateways that translate one directory access protocol into another	The NIS/LDAP gateway presented in Chapter 6
"How can users in a non-Unix administrative domain access services on Unix hosts?"	Cross-platform authentication services	Authenticating non-Microsoft clients against an active directory
"How can I join information contained in different directories?"	Distributed, multivendor directories glued together by referrals and references	Connecting directories from different vendors into a single DIT
"How can I unify access to the databases and directories held by multiple departments in my organization?"	Metadirectories that provide an integrated view of several disjointed directories and databases	Using an LDAP proxy server to translate entries from a second directory into the format needed by client applications
"How can I implement replication or synchronization between directories from different vendors?"	Push/pull agents that synchronize information from one directory to another	Customizing scripts or in-house tools that suck data from one server and uploading it to another directory after translating it into a format understood by the second server

This chapter examines ways to implement each approach. No single approach is a solution in and of itself; they're tools that you can use to assemble a solution that works in your environment. My intent, therefore, is to spur your imagination and introduce you to the different types of glue that are available for coordinating directory services.

9.2 Directory Gateways

Gateways are not a new concept; we've seen gateways between different email formats, different network filesystems, and so on for years. When building a gateway for directory services, one directory protocol is used as the frontend (the "face" presented to application clients). Another protocol is used between the gateway and the backend storage mechanism. The irony of using a directory gateway to unify access to an LDAP server is that LDAP itself was originally designed as a gateway protocol for X.500.

PADL's ypldapd daemon, presented in Chapter 6, is an example of a gateway between NIS and LDAP. Packages such as *ypldapd* tend to do one thing and do it well. In many respects, such a gateway can simply be viewed as another LDAP client. The gateway consumes LDAP information and makes that information available to its clients through another protocol.

Another example of an NIS/LDAP gateway is the NIS service distributed with Microsoft's "Windows Services for Unix (SFU)." This Active Directory add-on provides tools for importing data from a NIS domain into Active Directory. Once NIS data has been incorporated into Active Directory, SFU can provide services for NIS clients from the Active Directory domain. For more information on the SFU product, see http://www.microsoft.com/windows/sfu/.

The main advantage of using a gateway is that you usually don't have to modify any clients. This alone can save a great deal in the cost of administration. The disadvantage of using a gateway is that translating requests and replies from one protocol to the other requires additional overhead. Furthermore, clients can't take full advantage of the LDAP directory service; they're limited to the services offered by the gateway. In many environments, these disadvantages are relatively minor.

9.3 Cross-Platform Authentication Services

Cross-platform authentication is a term heard most often in IT departments that want to authenticate logons to Unix services using Microsoft's Active Directory,[1] or authenticate logons to Windows clients using a Unix-based LDAP server. In this scenario, we're not interested in interoperability between directory services, but between a specific directory service and nonnative clients (for example, Active Directory and Unix clients).

[1] Active Directory can be described as a network operating system (NOS) directory service that uses LDAPv3 as its primary access protocol and is, along with Kerberos 5, the major piece of Microsoft's larger domain infrastructure model. So while it is possible to use Active Directory as a vanilla LDAP directory service, I have never encountered a network that used Active Directory without a specific need for integration with other Microsoft technologies. More information about Active Directory can be found at http://www.microsoft.com/ad and in Windows 2000 Active Directory Services (O'Reilly).

Cross-platform authentication is the Holy Grail for many administrators, not just those dealing with Microsoft operating systems. Novell's eDirectory (formally called NDS) is available on a variety of platforms, including Windows, Linux, and Solaris. Novell provides tools such as a PAM module for NDS to integrate host authentication services with their directory. However, while Microsoft does provide some tools and sample source code for integrating Unix clients into an Active Directory domain (http://msdn.microsoft.com/library/en-us/dnactdir/html/kerberossamp.asp), there is currently no way to implement an Active Directory domain using non-Microsoft servers and technologies.

In all fairness, Microsoft's small offering of packages for Unix servers does not prevent Unix clients from using the user and group account information stored in an Active Directory domain. There are at least three methods for using Active Directory to authenticate Unix requests:

- The NIS/Active Directory gateway included in Microsoft's "Services for UNIX" package allows Unix clients to access information stored in Active Directory. We discussed this product briefly in the previous section.
- PADL's PAM and NSS LDAP libraries can act as a proxy server between the Unix services and Active Directory. The modules map attributes and object classes stored in Active Directory to something more suitable for consumption by Unix applications.
- Active Directory domains use Kerberos 5 for authenticating users. Interoperability between the implementations of Kerberos on Windows and other platforms is better than you might expect, but perhaps not as good as you would hope.

The remainder of this section examines the PAM/NSS solution in depth. At the end of this section, we'll discuss how to use Kerberos to enhance interoperability between OpenLDAP and Active Directory. The examples use a single Active Directory domain with the name *ad*.plainjoe.org using the default options provided by the *dcpromo* installation process. The domain name implies that the domain naming context is dc=ad,dc=plainjoe,dc=org.

Chapter 6 covered how to install and configure the PADL libraries with an OpenLDAP server supporting the RFC

9.4 Distributed, Multivendor Directories

Standard protocols go a long way to promote interoperability. While the schema for representing an LDAP referral can vary from vendor to vendor, the method of returning referral information to clients is defined as part of the core LDAPv3 protocol. This means that LDAP servers from various vendors can be linked into a single, logical, distributed directory.

But why go through all the trouble of building a multivendor directory? Why not settle on a single LDAP vendor, who has no doubt made it easy to build distributed directories by developing schemas to represent referrals and solving other problems that aren't addressed by the standards? And, as I've said elsewhere, we shouldn't use technologies just because they're there; if LDAP doesn't make life easier for us as administrators, and for the users of our systems, there's little point going through the effort of setting up an LDAP directory at all, let alone a distributed, multivendor directory.

However, sooner or later a single-vendor directory will force you to make decisions that you're uncomfortable with. Let's assume that you're adding a new application server, such as a calendaring system, at your site. This server is backed by an LDAP directory and requires certain protocol extensions from the directory. Naturally, the vendor has tested the application server with a particular LDAP server in mind—perhaps the vendor even sells an LDAP product (which, of course, is guaranteed to work with the calendar server). But as fate would have it, you've already invested a lot of time and effort in building an LDAP directory, and the directory server that supports the calendar server is not the directory server you've spent so much effort deploying. In this case, there are three possible solutions:

- Abandon the calendar server, since it is not supported by your existing LDAP server. However, you're probably installing the server because management wants you to do so; saying no probably isn't an option.
- Replace the existing directory with one that supports the calendar service. This solution probably doesn't involve throwing out all the work you did getting your directory service running—but you will have to redo a lot of it. And what happens the next time you're told to install an application that talks to an LDAP server? Will it be compatible with the server you've installed for the sake of the calendar service? It's clear that this isn't really an option, unless you want to spend your career playing "musical servers."
- Install a new LDAP server that supports the calendar application and include it as a subtree of your existing directory framework.

The last option is really the only option that makes sense. It allows you to augment, rather than replace, the directory infrastructure you've already built. Furthermore, sooner or later you will be forced to incorporate different LDAP servers into your network. The goal of standardization is to enable clients developed by one company to access servers developed by another; and even if this is presently a goal rather than a reality, your life will be easier if you work with this goal in mind.

The addition of a new vendor-dependent, LDAP-enabled application raises an important question: how is this solution any different than the myriad of application-specific directories of the past? The difference here is that there is a single access protocol for all clients and administration tools. The LDAP protocol is the unifying factor. While you still have applications that can talk only to a particular vendor's server, the common LDAP protocol allows you to

9.5 Metadirectories

The term *metadirectory* describes just about any solution that joins distinct, isolated data sources into a single logical volume. There are several popular metadirectory products on the market:

- MaXware MetaCenter (http://www.maxware.com/)
- Siemens DirXmetahub (http://www.siemens.ie/fixedoperators/CarrierNetworks/Meta/dirxmetahub.htm)
- Sun Microsystems SunOne (http://wwws.sun.com/software/products/meta_directory/home_meta_dir.html)
- Novell's eDirectory and DirXML combination (http://www.novell.com/products/edirectory/)
- Microsoft Metadirectory Services (http://www.microsoft.com/windows2000/technologies/directory/MMS)

For the sake of this section, we'll assume that a metadirectory is any directory service that presents an alternate view of a data source. OpenLDAP's proxy backend provides a simple means of translating one directory server's schema into a different view, suitable for particular client applications. There is no replication or synchronization of data because the proxy provides only an alternate view of the target directory; the OpenLDAP server providing the proxy doesn't actually store the data.

Imagine an email client that expects a directory service to provide an email address using the mail attribute type. Now consider that every user in an Active Directory domain is automatically assigned a Kerberos principal name of the form username@domain. If the email domain is configured so that each user's email address and Kerberos principal name (userPrincipalName) are synchronized (perhaps using an LDAP proxy service), then it makes no sense to duplicate this information just to provide a directory-based address book for a picky email application.

This scenario glosses over some details, such as where the mail domain stores email addresses.

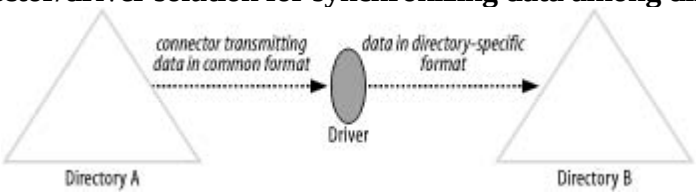
Before you can successfully create a proxy server, the Active Directory domain must meet the following requirements:

- The Active Directory domain must be configured for the DNS domain ad.plainjoe.org.
- The DNS name ad.plainjoe.org must resolve to the IP address of an Active Directory domain controller for

9.6 Push/Pull Agents for Directory Synchronization

Push/pull agents are common tools for synchronizing information between directories. In this case, a single agent manually pulls information from one directory service and massages the data to make it acceptable for upload to another directory server. Several directory vendors provide synchronization agents of this type in the form of connectors and drivers. A connector transfers data from one directory to another (see Figure 9-8) using a common format, often XML-based, while a driver translates the connector's data format to something understood by the local directory.

Figure 9-8. Using a connector/driver solution for synchronizing data among different directory services



A partial list of commercial connector/driver offerings includes:

- SunOne's XMLDAP (http://wwws.sun.com/software/products/directory_srvr/)
- Novell's DirXML (http://www.novell.com/products/edirectory/dirxml/)

The advantage that most commercial connector/driver solutions enjoy over in-house solutions is an inherent knowledge of when data changes in the directory. This means that the directory can trigger the connector upon any relevant change; in most cases, an external agent can detect a change only by polling the directory.

Despite this disadvantage, home-grown tools that act as middlemen between directory services can be very useful. The next chapter focuses on how to script directory operations using Perl and the Net::LDAP module.

9.6.1 The Directory Services Markup Language

The Extensible Markup Language (XML) has been hyped as the next big thing for several years now. Whether or not it has achieved its promise is a question I won't get into. LDAP has not been immune to the XML fever. The Directory Services Markup Language (DSML) is an XML schema for representing LDAP information using document fragments. DSML v1.0 could really only be described as an attempt to replace LDIF. With Version 2.0, however, released in May of 2002, DSML has grown up and gained some new and interesting functionality.[4]

[4] The latest information about DSML can be found at the OASIS Directory Services Technical Committee's web page (http://www.oasis-open.org/committees/dsml/).

DSMLv2 is designed to provide methods for representing LDAP queries, updates, and the responses to these operations in XML. This means that it would be possible for small, embedded devices to access LDAP services without relying on an LDAP client library; they only need the ability to parse XML. Because XML-based standards such as SOAP will only become more prevalent over time, the Oasis Directory Service TC has included a

Chapter 10. Net::LDAP and Perl

No book on system administration is complete without some coverage of scripting. For many administrators, the scripting language of choice is Perl. Perl is very good at dealing with text files (such as LDIF files), and many third-party modules make it easy to accomplish complex tasks.[1]

[1] For more information on Perl, visit the O'Reilly Perl web site at http://www.perl.com/ or the Perl Monger's web site at http://www.perl.org/. If you're new to Perl, I recommend Learning Perl, by Randall Schwartz and Tom Phoenix (O'Reilly) and Programming Perl, by Larry Wall, Tom Christiansen, and Randall Schwartz (O'Reilly).

This chapter doesn't cover the basics of Perl programming. I assume that you are already comfortable with the language and its fundamental concepts, such as regular expressions, but none of the examples will require the help of a Perl guru for interpretation. Note that the scripts in this chapter are generally lax about conventions used in production Perl code, such as the use strict pragma and variable scoping (e.g., my or local).

10.1 The Net::LDAP Module

Two widely distributed Perl modules make it easy to write scripts that interact with an LDAP directory. One of these is the PerLDAP module, written by Leif Hedstrom from Netscape Communications (http://www.mozilla.org/directory/perldap.html). However, the last version was released in October of 2000.

A more active project, and the module that I discuss in this chapter, is Graham Barr's perl-ldap module (often referred to as Net::LDAP). The examples in this chapter are based on Version 0.26 of this module. The module's home is located at http://perl-ldap.sourceforge.net/, but it's simpler to get it through the Comprehensive Perl Archive Network (CPAN) at http://search.cpan.org. Before you install Net::LDAP, make sure that the following modules are present:

URI

If you want to parse ldap:// URIs

Digest::MD5

For Base64 encoding

IO::Socket::SSL

For LDAPS and StartTLS support

XML::Parser

To read and write DSML files

Authen::SASL

For SASL authentication support

All of these modules (and any of their requisite modules) can be downloaded from CPAN mirrors. As a convenience, several of these modules have been packaged into a single module named Bundle::Net::LDAP, which can also be download from CPAN.

One of the easiest ways to ensure that all dependencies for a Perl module are met is to use the interactive shell provided by Andreas Koenig's CPAN module. After downloading and installing this module from http://search.cpan.org/search?dist=CPAN, you can learn about its features by executing the command *perldoc CPAN*.

Programming with the Net::LDAP module is not tricky. You can discover a lot about it by typing the command perldoc Net::LDAP at a shell prompt; additional documentation can be found under Net::LDAP::Examples and Net::LDAP::FAQ.

10.2 Connecting, Binding, and Searching

To get started with the Net::LDAP module, we will write a basic LDAP query script named *search.pl*. This script illustrates the methods used to connect to a directory server and retrieve information. It begins by importing the Net::LDAP symbols via the use pragma:

```
#!/usr/bin/perl
use Net::LDAP;
```

After the module has been included, you can create a new instance of a Net::LDAP object. To create a new Net::LDAP instance, you need the hostname of the LDAP server to which the script should connect. The constructor allows several optional arguments, of which the most common and useful are:

port

The TCP port on which the directory server is listening. If this parameter is not defined, it defaults to the well-known LDAP port (389).

version

The LDAP version to be used when connecting to the server. The default is Version 2 in the 0.26 release. However, this is likely to change in the future. Always explicitly set the version parameter if your Perl program replies with LDAPv3 features (such as SASL or referrals).

timeout

The time in seconds that the module should wait when contacting the directory server. The default value of 120 seconds is sufficient for most situations, but for more complex searches or when communicating with a very large directory, it may be necessary to increase this value.

The next line of code establishes a connection to the host ldap.plainjoe.org on port 389 using Version 3 of the protocol. The returned value is a handle to a Net::LDAP object that can be used to retrieve and modify data in the directory.

```
$ldap = Net::LDAP->new ("ldap.plainjoe.org", port =>389,
                        version => 3 );
```

The script can bind to the directory after it obtains a handle to the LDAP server. By default, Net::LDAP uses an implicit anonymous bind, but it supports all the standard binds defined by the LDAPv3 RFCs (anonymous, simple, and SASL). For now, we only examine how to use a simple bind.

However, before binding to the server, call start_tls( ) to encrypt the connection; you don't want to send the user DN and password across the network in clear text. In its simplest form, the start_tls( ) method requires no parameters and appears as:

```
$ldap->start_tls(  );
```

Checking for Errors

Most of the Net::LDAP methods return an object with two methods for obtaining the function's return status. The code( ) method retrieves the integer return value from the method call that created the object, and the error( ) method returns a descriptive character string associated with the numeric code. The constants for the various LDAP errors are contained in the Net::LDAP::Constant module. Specific error codes can be included in your code by adding a line similar to the following one:

10.3 Working with Net::LDAP::LDIF

The search.pl script provided a simple introduction to retrieving data from an LDAP directory. However, the query results represented the state of the directory at a single point in time. The script has no good way to save the search results, and the way in which it prints the information is useful for humans, but not useful to any other LDAP tools. You need the ability to save the results in a format that can be parsed by other LDAP tools: in other words, you need to be able to read and write LDIF files directly from Perl code.

The Net::LDAP::LDIF module provides the ability to work with LDIF files. To introduce Net::LDAP::LDIF, we'll revisit search.pl and replace the call to dump( ) with code to produce valid LDIF output. Your first modification to the script is to add a second use pragma that imports the LDIF module:

```
use Net::LDAP::LDIF;
```

Next, the script must create a new instance of a Net::LDAP::LDIF object. Output from this object can be linked to an existing file handle such as STDOUT, as shown here:

```
$ldif = Net::LDAP::LDIF->new (STDOUT, "w")
     or die $!;
```

It is possible to pass a filename to the new( ) method, as well as inform the module how this file will be used ("r" for read, "w" for write + truncate, and "a" for write + append). This line of code creates an LDIF output stream named *result.ldif* in the current directory:

```
$ldif = Net::LDAP::LDIF->new ("./result.ldif", "w")
     or die $!;
```

It is best to use this code after you've run the search and confirmed that it produced some results. So, you open the file after the script has tested that $msg->count( ) > 0:

```
if ( $msg->count(  ) > 0 ) {
    print $msg->count(  ), " entries returned.\n";

    $ldif = Net::LDAP::LDIF->new (scalar<STDOUT>, "w")
          or die $!;
```

Finally, replace the entire foreach loop that calls dump( ) on each entry with a single call to the write_entry( ) method of Net::LDAP::LDIF:

```
     $ldif->write_entry($msg->all_entries(  ));
```

write_entry( ) accepts either a single Net::LDAP::Entry or a one-dimensional array of these objects. The new loop is:

```
if ( $msg->count(  ) > 0 ) {
    print $msg->count(  ), " entries returned.\n";

    $ldif = Net::LDAP::LDIF->new (scalar<STDOUT>, "w")
          or die $!;

    $ldif->write_entry($msg->all_entries(  ));
}
```

Now the output of the script looks like this:

```
dn: cn=Gerald Carter,ou=contacts,dc=plainjoe,dc=org
cn: Gerald Carter
mail: jerry@samba.org
```

This doesn't look like a big change, but it's an important one. Because the data is now in LDIF format, other tools such as *ldapmodify* can parse your script's output.

10.4 Updating the Directory

Searching for objects in the directory is only the beginning. The real power of scripting is that it allows you to modify the directory; you can add entries, delete entries, and modify existing entries.

10.4.1 Adding New Entries

The first script, import.pl, reads the contents of an LDIF file (specified as a command-line argument) and adds each entry in the file to the directory. Here's a starting point; it resembles the last version of your search.pl script:

```
#!/usr/bin/perl
##
## Usage: ./import.pl filename
##
## Author: Gerald Carter <jerry@plainjoe.org>
##
use Net::LDAP;
use Net::LDAP::LDIF;

## Connect and bind to the server.
$ldap = Net::LDAP->new ("ldap.plainjoe.org", port =>389,
                        version => 3 )
or die $!;

## Secure data and credentials.
$result = $ldap->start_tls(  );
die $result->error(  ) if $result->code(  );

## Bind to the server. The account must have sufficient privileges because you will
## be adding new entries.
$result = $ldap->bind(
        "cn=Directory Admin,ou=people,dc=plainjoe,dc=org",
         password => "secret");
die $result->error(  ) if $result->code(  );

## Open the LDIF file or fail. Check for existence first.
die "$ARGV[0] not found!\n" unless ( -f $ARGV[0] );
$ldif = Net::LDAP::LDIF->new ($ARGV[0], "r")
      or die $!;
```

Once the script has a handle to the input file, you can begin processing the entries. Net::LDAP::LDIF has an eof( ) method for detecting the end of input. The main loop continues until this check returns true.

```
while ( ! $ldif->eof ) {
    ## Get next entry and process input here.
}
```

Retrieving the next LDIF entry in the file is extremely easy because the Net::LDAP::LDIF module does all the work, including testing the file to ensure that its syntax is correct. If the next entry in the file is valid, the read_entry( ) method returns it as a Net::LDAP::Entry object.

```
$entry = $ldif->read_entry(  );
```

If the call to read_entry( ) fails, you can retrieve the offending line by invoking the error_lines( ) routine:

```
if ( $ldif->error(  ) ) {
    print "Error msg: ", $ldif->error(  ), "\n";
    print "Error lines:\n", $ldif->error_lines(  ), "\n";
    next;
}
```

If no errors occur, the script adds the entry it has read from the file to the directory by invoking the Net::LDAP add( )

10.5 Advanced Net::LDAP Scripting

At this point, we've covered all the basics: binding to a server, reading, writing, and modifying entries. The remainder of the chapter covers more advanced programming techniques. We'll start by discussing how to handle referrals and references returned from a search operation.

10.5.1 References and Referrals

It's important for both software developers and administrators to understand the difference between a reference and a referral. These terms are often confused, probably because the term "referral" is overused or misused. As defined in RFC 2251, an LDAP server returns a reference when a search request cannot be completed without the help of another directory server. I have called this reference a "subordinate knowledge reference" earlier in this book. In contrast, a referral is issued when the server cannot service the request at all and instead points the client to another directory that may have more knowledge about the base search suffix. I have called this link a "superior knowledge reference" because it points the client to a directory server that has superior knowledge, compared to the present LDAP server. These knowledge references will be returned only if the client has connected to the server using LDAPv3; they aren't defined by LDAPv2.

A Net::LDAP search returns a Net::LDAP::Reference object if the search can't be completed, but must be continued on another server. In this case, the reference is returned along with Net::LDAP::Entry objects. If a search requires a referral, it doesn't return any Entry objects, but instead issues the LDAP_REFERRAL return code. Both references and referrals are returned in the form of an LDAP URL. To illustrate these new concepts and their use, we will now modify the original search.pl script to follow both types of redirection. As of Version 0.26, the Net::LDAP module does not help you follow references or referrals—you have to do this yourself.

To aid in parsing an LDAP URL, use the URI::ldap module. If the URI module is not installed on your system, you can obtain it from http://search.cpan.org/. LDAP_REFERRAL is a constant from Net::LDAP::Constant that lets you check return codes from the Net::LDAP search( ) method.

```
#!/usr/bin/perl
## Usage: ./fullsearch.pl name
##
## Author: Gerald Carter <jerry@plainjoe.org>
##
use Net::LDAP qw(LDAP_REFERRAL);
use URI::ldap;
```

The script then connects to the directory server:

```
$ldap = Net::LDAP->new ("ldap.plainjoe.org",
                  port => 389,
                  version => 3 )
or die $!;
```

To simplify the example, we will omit the bind( ) call (from the original version of search.pl) and bind to the directory anonymously. We'll also request all attributes for an entry rather than just the cn and mail values. The callback parameter is new. Its value is a reference to the subroutine that should process each entry or reference returned by the search:

```
$msg = $ldap->search(
                base => "ou=people,dc=plainjoe,dc=org",
                scope => "sub",
                filter => "(cn=$ARGV[0])",
                callback => \&ProcessSearch );

ProcessReferral( $msg->referrals(  ) )
```

Part III: Appendixes

Appendix A. PAM and NSS

A.1 Pluggable Authentication Modules

The concept of Pluggable Authentication Modules (PAM) was first designed by Sun Microsystems' SunSoft development group and is defined in the Open Software Foundations RFC 86.0. PAM provides a framework that allows vendors and administrators to customize the services used to authenticate users on a local computer system. For example, logging onto the console of a system may require stronger authentication than logging into a host across the network via ssh. Configuring systems to use different PAM modules (e.g., smart cards or passwords) for different services (e.g., login or ssh) allows administrators to implement as much or as little security as the systems require.

In practice, administrators are exposed to this framework through shared libraries that implement various security, accounting, or account management policies. On Linux and Solaris systems, you can list the installed PAM modules by examining the contents of */lib/security*. The most commonly used module for normal lookups in the system list of accounts (including */etc/shadow*) is the pam_unix.so library. Linux's PAM implementation includes a drop-in replacement module named pam_pwdb.so, which relies on the generic interface to the Password Database library (http://www.kernel.org/pub/linux/libs/pam/modules.html).

PAM is a framework for authentication and authorization. Authentication is the process of proving you are who you say you are, while authorization is the process of determining what you are allowed do, given that you have established your identity. Applications can query the PAM interface to ask questions about a user or to inform a PAM module of a particular event. For example, "Does this password match with this login name?", "Is this user allowed to log onto this host at 10 p.m. on a Saturday?", "The user named *smitty* logged onto the local system on Thu Dec 19 21:04:27 CST 2002," or "Change this user's password to secret." In each case, the application uses a specific module to process each type of questions or event that can occur during the logon process.

A.1.1 Configuring PAM

PAM configuration files follow one of two formats. In modern Linux distributions, each application or service that possesses an individual configuration file is located in the directory /etc/pam.d/. Each file is usually named after the type of service it controls. For example, Qualcomm's Qpopper server, a POP3 daemon, uses the PAM file */etc/pam.d/pop3*, and the *login* service is configured by the file /etc/pam.d/login. Here's a valid configuration for a PAM-enabled login service:

```
 ## /etc/pam.d/login
## Log in using entries from /etc/[password|shadow].
auth       required   /lib/security/pam_unix.so
## Allow root logons only from a tty listed in /etc/securetty.
auth       required   /lib/security/pam_securetty.so
## Don't allow logins (except root) if /etc/nologin exists.
auth       required   /lib/security/pam_nologin.so

## Ensure that account and password are active and have not expired.
account    required   /lib/security/pam_unix.so

## Log username via syslog.
session    required   /lib/security/pam_unix.so

## Enforce good passwords.
password   required   /lib/security/pam_cracklib.so
## Change the password in /etc/[password|shadow].
password   required   /lib/security/pam_unix.so
```

The older type of PAM configuration file, still supported on Solaris, places information for all services in one file, */etc/pam.conf*. In the absence of the */etc/pam.d/* directory, the Linux-PAM implementation will fall back to using *pam.conf*. This file is similar to the newer PAM configuration file, except that the first entry on each line is the name of the service being configured. In newer configuration files (such as the login file listed above), the name of the

A.2 Name Service Switch (NSS)

The Name Service Switch (NSS) framework was designed to let administrators specify which files or directory services to query to obtain information. For example, it's frequently used to specify whether a system should perform hostname lookups in */etc/hosts*, NIS, or DNS. Here's an entry from a typical NSS configuration file, named */etc/nsswitch.conf*. It instructs the local machine to check its own */etc/hosts* file first and to consult DNS only if the entry is not located. NIS is not consulted at all.

```
hosts:      files dns
```

NSS can provide similar services for many different administrative databases. The following databases are generally defined in */etc/nsswitch.conf*:

passwd shadow group hosts ethers networks protocols rpc services netgroup aliases automount

You can configure a different lookup method for each database. An NSS module does not need to support all of the databases listed above. Some lookup modules support only user accounts. The libnss_dns.so library is designed to resolve only hostnames and network addresses.

A typical NSS configuration for an LDAP-enabled host would appear as:

```
# /etc/nsswitch.conf
# Legal entries are:
#
# nisplus or nis+: Use NIS+ (NIS Version 3)
# nis or yp: Use NIS (NIS Version 2)
# dns: Use DNS (Domain Name Service)
# files: Use the local files
# db: Use the local database (.db) files
# compat: Use NIS on compat mode
# hesiod: Use Hesiod for user lookups
# ldap: Use PADL's nss_ldap

## How to handle users and groups
passwd:     files ldap
shadow:     files ldap
group:      files ldap

## DNS should be authoritative; use files only when DNS is not available.
hosts:      dns [NOTFOUND=return] files

bootparams: ldap files

ethers:     ldap files
netmasks:   ldap files
networks:   ldap files
protocols:  ldap files
rpc:        ldap files
services:   ldap files

netgroup:   files ldap
automount:  files ldap
aliases:    files
```

More information can be found on the *nsswitch.conf(5)* manpage.

Appendix B. OpenLDAP Command-Line Tools

B.1 Debugging Options

Most OpenLDAP tools provide an option for setting the log level during execution. Table B-1 lists the information recorded with each level. Log levels are additive, so a log level of 24 means to print packets sent and received as well as logging all connection management functions.

Table B-1. OpenLDAP logging levels

Level	**Information recorded**
-1	All logging information
0	No logging information
1	Trace function calls
2	Packet-handling debugging information
4	Heavy trace debugging
8	Connection management
16	Packets sent and received
32	Search filter processing
64	Configuration file processing
128	Access control list processing
256	Statistics for connection, operations, and results
512	Statistics for results returned to clients
1024	Communication with shell backends

B.2 Slap Tools

The collection of slap tools included with OpenLDAP are provided to import and export data directly from the DB files used for supporting an OpenLDAP server.

B.2.1 slapadd(8c)

This tool reads LDIF entries from a file or standard input and writes the new records to a *slapd* database (see Table B-2).

Table B-2. Summary of slapadd command-line arguments

Option	**Description**
-c	Continues processing input in the event of errors.
-b suffix-n integer	Specify which database in the configuration file to use by the directory's suffix (-b) or by its location (-n) in the *slapd.conf* file (the first database listed is numbered 0). These options are mutually exclusive.
-d integer	Specifies which debugging information to log. See the loglevel parameter in slapd.conf for a listing of log levels.
-f filename	Specifies which configuration file to read.
-l filename	Specifies the LDIF file to use for input. In the absence of this option, *slapadd* reads data from standard input.
-v	Enables verbose mode.

B.2.2 slapcat(8c)

This tool reads records from a *slapd* database and writes them to a file or standard output (see Table B-3).

Table B-3. Summary of slapcat command-line arguments

Option	**Description**
-c	Continues processing input in the event of errors.

B.3 LDAP Tools

OpenLDAP's set of LDAP client tools can be used to communicate with any LDAPv3 server (see Table B-6).

Table B-6. Command-line options common to ldapsearch, ldapcompare, ldapadd, ldapdelete, ldapmodify, and ldapmodrdn

Option	**Description**
-d integer	Specifies what debugging information to log. See the loglevel slapd.conf parameter for a listing of log levels.
-D binddn	Specifies the DN to use for binding to the LDAP server.
-e [!]ctrl[=ctrlparam]	Defines an LDAP control to be used on the current operation. See also the -M option for the manageDSAit control.
-f filename	Specifies the file containing the LDIF entries to be used in the operations.
-H URI	Defines the LDAP URI to be used in the connection request.
-I	Enables the SASL "interactive" mode. By default, the client prompts for information only when necessary.
-k	Enables Kerberos 4 authentication.
-K	Enables only the first step of the Kerberos 4 bind for authentication.
-M-MM	Enable the Manager DSA IT control. This option is necessary when modifying an entry that is a referral or an alias. -MM requires that the Manager DSA IT control be supported by the server.
-n	Does not perform the search; just displays what would be done.

Appendix C. Common Attributes and Objects

This appendix is provided as a quick reference for schema items used throughout this book. It is by no means a complete set of attributes and object classes that you may encounter in the wild. The schema items not listed here should not be assumed to be less important or less commonly used. These are just the primary ones I have focused on in the examples.

C.1 Schema Files

Table C-1 tells you where you can find schema files.

Table C-1. Where to find schema files

Software	**Schema files included**
Bind 9 (schema file located at http://www.venaas.no/ldap/bind-sdb/)	*dnszone.schema*
LDAP System Administration (http://www.oreilly.com/catalog/ldapsa/)	*idpool.schema* *printer.schema*
OpenLDAP (http://www.openldap.org/)	*core.schema* *corba.schema* *cosine.schema* *inetorgperson.schema* *java.schema* *misc.schema* *nis.schema* *openldap.schema*
Samba (http://www.samba.org/)	*samba.schema*
Sendmail (http://www.sendmail.org/)	*sendmail.schema*
FreeRadius (http://www.freeradius.org)	*RADIUS-LDAPv3.schema*

C.2 Attributes

Table C-2 outlines some common attributes presented in this book.

Table C-2. Common attributes presented in this book

Name	Single value	Description
cn		Common name of entity
dc		Single domain component of an FQDN
displayName	✓	Preferred name to use when displaying entry
gidNumber	✓	Numeric Unix group ID
givenName		First name by which an entity is known
mail		Email address represented as an RFC 822 mailbox
ou		organizationalUnit to which this entry belongs
sn		Last name by which an entity is known
telephoneNumber		Telephone number (supports international dialing format)
uid		Login name for a user account
uidNumber	✓	Numeric Unix user ID
userPassword		Password asssociated with an entry

C.3 Object Classes

This section describes some object classes presented in this book.

account *(cosine.schema)*

Type

STRUCTURAL

Parent

top

Attributes

Mandatory: uid

Optional: description, seeAlso, localityName, organizationName, organizationalUnitName, host

dcObject *(core.schema)*

Type

AUXILIARY

Parent

top

Attributes

Mandatory: dc

Optional: None

dNSZone *(dnszone.schema)*

Appendix D. LDAP RFCs, Internet-Drafts, and Mailing Lists

D.1 Requests for Comments

RFC documents are available online at http://www.rfc-editor.org/. The list here includes LDAPv3-related RFCs in numerical order.

RFC 1274

"The COSINE and Internet X.500 Schema". P. Barker and S. Kille. November 1991. Status: Proposed Standard.

RFC 2079

"Definition of an X.500 Attribute Type and an Object Class to Hold Uniform Resource Identifiers (URIs)". M. Smith. January 1997. Status: Proposed Standard.

RFC 2247

"Using Domains in LDAP/X.500 Distinguished Names". S. Kille et al. January 1998. Status: Proposed Standard.

RFC 2251

"Lightweight Directory Access Protocol (v3)". M. Wahl, T. Howes, and S. Kille. December 1997. Status: Proposed Standard.

RFC 2252

"Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions". M. Wahl et al. December 1997. Status: Proposed Standard.

RFC 2253

"Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names". M. Wahl, S. Kille, and T. Howes. December 1997. Status: Proposed Standard.

RFC 2254

"The String Representation of LDAP Search Filters". T. Howes. December 1997. Status: Proposed Standard.

RFC 2255

"The LDAP URL Format". T. Howes and M. Smith. December 1997. Status: Proposed Standard.

RFC 2256

"A Summary of the X.500(96) User Schema for use with LDAPv3". M. Wahl. December 1997. Status: Proposed Standard.

RFC 2293

"Representing Tables and Subtrees in the X.500 Directory". S. Kille. March 1998. Status: Proposed Standard.

RFC 2294

"Representing the O/R Address Hierarchy in the X.500 Directory Information Tree". S. Kille. March 1998. Status: Proposed Standard.

RFC 2307

"An Approach for Using LDAP as a Network Information Service". L. Howard. March 1998. Status: Experimental.

RFC 2377

"Naming Plan for Internet Directory-Enabled Applications". A. Grimstad et al. September 1998. Status: Informational.

RFC 2589

"Lightweight Directory Access Protocol (v3): Extensions for Dynamic Directory Services". Y. Yaacovi, M. Wahl, and T. Genovese. May 1999. Status: Proposed Standard.

D.2 Mailing Lists

OpenLDAP.org hosts several public mailing lists, all of which are described at http://www.openldap.org/lists/. The two most frequented lists are *openldap-software* (discussions about software created as part of the OpenLDAP project) and *openldap-devel* (technical discussions relating to OpenLDAP development). You can subscribe to a list by sending an email to openldap-<list>-request@OpenLDAP.org, in which <list> is either software or devel, with the word "subscribe" in the body of the message.

The University of Michigan hosts a general LDAP mailing list. You can subscribe to its list by sending email to ldap-request@umich.edu with the word "subscribe" as the subject or by accessing the web interface found at http://listserver.itd.umich.edu/.

Appendix E. slapd.conf ACLs

This appendix is provided as a quick reference to the access control rule syntax used in *slapd.conf*. The general syntax of an access control rule is:

```
access to what {by who how-much [control]}+
```

Three syntax items are referred to frequently in the tables found in this appendix:

dnstyle

Can be one of [regex | base | one | subtree | children]

style

Can be one of [regex | base]

regex

Will be expanded as described by the *regex(7)* manpage

E.1 What?

Table E-1 presents a summary of access rule targets.

Table E-1. Summary of access rule targets

What?	Description
*	Everything
dn[.*dnstyle*]=*regex*	The entries specified by the *style* beginning at the suffix *regex*
filter=*ldapfilter*	The entries returned by applying the RFC 2254 LDAP filter to the directory
attrs=*attribute_list*	The list of attributes specified

E.2 Who?

Table E-2 presents a summary of access rule entities.

Table E-2. Summary of access rule entities

Who?	**Description**
*	Everyone (including anonymous connections)
anonymous	Non-authenticated connections
users	Authenticated connections
self	The user represented by the DN of the target entry
dn[*dnstyle*]=*regex*	The user represented by the specified DN.
dnattr=*attribute_name*	The user represented by the DN stored in the specified attribute in the target entry
group[/*obj*[/*attr*]][.*style*]=*pattern*	The members of the group represented by *pattern*
peername[.*style*]=*pattern* sockname[.*style*]=*pattern* domain[.*style*[,*modifier*]]=*pattern* sockurl[.*style*]=*pattern*	Host-/filesystem-based access mechanisms
ssf=*n* transport_ssf=*n* tls_ssf=*n*	Defined minimum security levels for access to be granted

E.3 How Much?

OpenLDAP supports two modes of defining access. The general form of the access specifier clause is:
`[self]{level|priv}`

The special modifier self implies special access to self-owned attributes such as the member attribute in a group.

While the access level model implements incremental access (higher access includes lower access levels), the privilege model requires that an administrator explicitly define access for each permission using the =, +, and - operators to reset, add, and remove permissions, respectively (see Table E-3).

Table E-3. Summary of access and privilege levels from most (top) to least (bottom)

Access level	**Privilege**	**Permission granted**
write	w	Access to update attribute values (e.g., change this telephoneNumber to 555-2345).
read	r	Access to read search results (e.g., Show me all the entries with a telephoneNumber of 555*).
search	s	Access to apply search filters (e.g., Are there any entries with a telephoneNumber of 555*?).
compare	c	Access to compare attributes (e.g., Is your telephoneNumber 555-1234?).
auth	x	Access to bind (authenticate). This requires that the client send a username in the form of a DN and some type of credentials to prove his or her identity.
none		No access.

Control flow from one access rule to the next can be managed by the keywords stop, continue, and break (see Table E-4).

Table E-4. Control flow keywords in access rules

E.4 Examples

Grant authenticated users the capability to read the cn attribute with the following:

```
access to attrs=cn
     by users read
```

Grant a single, specified user the capability to write to all posixAccount entries below the ou=people container with the following. This does not include permission to add new entries directly below ou=people.

```
access to dn.children="ou=people,dc=plainjoe,dc=org"
    filter=(objectclass=posixAccount)
    by dn="uid=admin,ou=people,dc=plainjoe,dc=org" write
```

Grant everyone the capability to attempt to authenticate against an entry's password with the following. The owner of the entry should also be given read and write access.

```
access to attrs=userPassword
   by * +x continue
   by self +rw
```

Restrict access to the administration organizational unit to members of the admin groupOfNames object with the following:

```
access to dn.subtree="ou=administration,dc=plainjoe,dc=org"
    by group/groupOfNames/member=
       "cn=admin,ou=group,dc=plainjoe,dc=org" write
    by * none
```

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of LDAP System Administration is a mink (Mustela vison). Mink are found throughout the United States and Canada except in Arizona, the Arctic, and some offshore islands. A mink's fur is mostly brown with some white spots around the throat, chin, and chest. Its coat is thick, soft, and waterproof (thanks to guard hairs covered with an oily protective substance). Its body is streamlined and skinny with short legs and an elongated face. As part of its water-loving nature, a mink's toes are partially webbed. Body length varies but is usually around two feet. The tail comprises almost half of a mink's total length.

Females become fertile during the winter and give birth in April or May. A typical litter ranges between one and eight offspring. M. vison is a solitary species; males are particularly intolerant of each other. They mark their territories with a pungent, musky secretion from their oversized anal glands. They are especially active at night and are skilled swimmers and climbers. Mink dig burrows in banks of lakes and rivers, or they may occupy abandoned dens of other mammals, such as muskrats. Their tastes in food changes from season to season, but they tend to dine on small mammals such as mice, rabbits, and shrews, along with fish and duck.

The main threat to the mink's existence continues to be the fur industry. Most U.S. states and all of Canada have limited trapping seasons with strict quotas on catch size. These provisions help keep mink population densities constant. Mink have few natural enemies other than humans. Occasionally, they will be hunted by coyotes, bobcats, and other meat-eaters.

Matt Hutchinson was the production editor and copyeditor for LDAP System Administration. Genevieve d'Entremont proofread the book. Genevieve d'Entremont, Emily Quill and Mary Anne Weeks Mayo provided quality control. Jamie Peppard provided production assistance. Julie Hawks wrote the index.

Emma Colby designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

Bret Kerr designed the interior layout, based on a series design by David Futato. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in this book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Matt Hutchinson.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.