CLASSIC NT HAS MANY ECCENTRICITIES—big and small—that limit its scalability and functionality. Many of these eccentricities stem from NT's clumsy, flat-file, Registry-based account management system. What is lacking in classic NT is a true *directory service* capable of handling the management chores for a network containing hundreds of thousands, if not millions, of users, computers, groups, printers, shared folders, network appliances, and so forth.

The hallmark of modern Windows is an enterprise-class directory service called Active Directory. We're going to spend the next six chapters learning to configure, deploy, manage, and fix Active Directory. The purpose of this chapter is to introduce you to the components of Active Directory and how they fit together. We'll also take an initial look at the tools provided by Microsoft to access and modify the contents of Active Directory.

New Features in Windows Server 2003

Microsoft has done quite a bit of tuning on Active Directory in Windows Server 2003 to improve scalability and speed and to correct a couple of key deficiencies. Some of these updates might not make much sense until you read further, but here is a synopsis to use for reference. The first three features require having Windows Server 2003 on every domain controller:

- **Site scalability.** The calculations for determining replication topology between sites have been streamlined. This corrects a problem where large organizations with hundreds of sites might experience replication failure because the topology calculations cannot be completed in the time allotted to them.

- **Backlink attribute replication.** Group members are now replicated as discrete entities instead of replicating the entire group membership list as a single unit. This corrects a problem where membership changes made to the same group on different domain controllers in the same replication interval overwrite each other.
- **Federations.** A new trust type called Forest was added to simplify transitive trust relationships between root domains in different forests. Using Forest trusts, it is possible to build a *federation* of independent Active Directory forests. This feature does not implement true "prune and graft" in Active Directory, but it goes a long way toward simplifying operations within affiliated organizations.
- **Simplified domain logon.** Universal group membership can be cached at non-global catalog servers. This permits users to log on even if connectivity to a global catalog server is lost. This enhancement is coupled with a feature in XP where the domain\name result of cracking a User Principal Name (UPN) is cached locally. This permits a user at an XP desktop to log on with the format `user@company.com` even if a global catalog server is not available.
- **Application naming contexts.** Windows Server 2003 introduces the capability to create new naming contexts to hold DNS record objects for Active Directory Integrated zones. One naming context holds domain zone records and one holds the _msdcs records used throughout a forest. These naming contexts make it possible to target replication of DNS zones only to domain controllers that are running DNS.
- **Eliminate piling onto new domain controllers.** There is potential for a problem when an NT4 primary domain controller (PDC) is upgraded to Windows Server 2003. In this circumstance, all existing Windows 2000 and XP desktops will use the newly promoted PDC as a logon server. In Windows Server 2003, domain controllers can be configured to respond to modern Windows clients as if they were still classic NT domain controllers until sufficient domain controllers are available to handle local authentication. This feature is also available in Windows 2000 SP2 and later.
- **DNS diagnostics.** Proper DNS configuration is critical for proper Active Directory operation. The Domain Controller promotion utility now performs a suite of DNS diagnostics to ensure that a suitable DNS server is available to register the service locator resource records associated with a Windows domain controller.

- **Fewer global catalog rebuilds.** Adding or removing an attribute from the Global Catalog no longer requires a complete synchronization cycle. This minimizes the replication traffic caused by adding an attribute to the GC.
- **Management console enhancements.** The Active Directory Users and Computers console now permits drag-and-drop move operations and modifying properties on multiple objects at the same time. There is also the capability of creating and storing custom LDAP queries to simplify managing large numbers of objects. The new MMC 2.0 console includes scripting support that can eliminate the need to use the console entirely.
- **Real-time LDAP.** Support was added for RFC 2589, "LDAPv3: Extensions for Dynamic Directory Services." This permits putting time-sensitive information in Active Directory, such as a user's current location. Dynamic entries automatically time out and are deleted if they are not refreshed.
- **Enhanced LDAP security.** Support was added for digest authentication as described in RFC 2829, "Authentication Methods for LDAP." This makes it easier to integrate Active Directory into non-Windows environments. Support was also added for RFC 2830, "LDAPv3: Extension for Transport Layer Security." This permits using secure connections when sending LDAP (Lightweight Directory Access Protocol) queries to a domain controller.
- **Schema enhancements.** The ability was added to associate an auxiliary schema class to individual objects rather than to an entire class of objects. This association can be dynamic, making it possible to temporarily assign new attributes to a specific object or objects. Attributes and object classes can also be declared defunct to simplify recovering from programming errors.
- **LDAP query enhancements.** The LDAP search mechanism was expanded to permit searching for individual entries in a multivalued Distinguished Name (DN) attribute. This is called an *Attribute Scoped Query*, or ASQ. For example, an ASQ could be used to quickly list every group to which a specific user belongs. Support was also added for Virtual List Views, a new LDAP control that permits large data sets to be viewed in order instead of paging through a random set of information. This change permits Windows Server 2003 to show alphabetically sorted lists of users and groups in pick lists.
- **Interoperability.** Support was added for RFC 2798, "Definition of the inetOrgPerson LDAP Object Class." This enhances interoperability with Netscape and NetWare directory services, both of which use the inetOrgPerson object class to create User objects.

- **Speedier domain controller promotions.** The capability was added for using a tape backup of the Active Directory database to populate the database on a new domain controller. This greatly simplifies domain controller deployments in situations where it is not practical to ship an entire server.
- **Scalability.** The maximum number of objects that can be stored in Active Directory was increased to over one billion.

Limitations of Classic NT Security

The first questions you may ask when hunkering down to study Active Directory is, "What is it?" and "Why have it?" This section answers the second question. The remainder of the chapter answers the first.

Account administration in a classic NT network is hampered by many limitations. The most important of these limitations are the following:

- Restricted SAM size
- Multiple logon IDs
- Single point of failure at the primary domain controller
- Poor operational performance
- Poor replication performance
- Lack of management granularity
- The fact that security databases differ between servers and domain controllers
- Nontransitive trust relationships

I'm going to discuss each of these limitations to show exactly how they hinder classic NT operations. This also helps to understand why certain decisions were made in the design of Active Directory.

Restricted Account Database Size

Security accounts in classic NT are stored in the Security Account Manager database, called the SAM for short. The SAM is a flat-file database consisting of a set of Groups and a set of Users. Computer accounts are also included in the SAM as a special form of user account.

SAM Database Structure

Ordinarily, you cannot view the contents of the SAM database because the Registry only permits access by the System account. If you want to take a peek inside, you can set the Registry permissions to give your account or the Administrators group Read access. Actual data is encrypted and stored in binary format, but you can view the structure. Figure 6.1 shows an example.

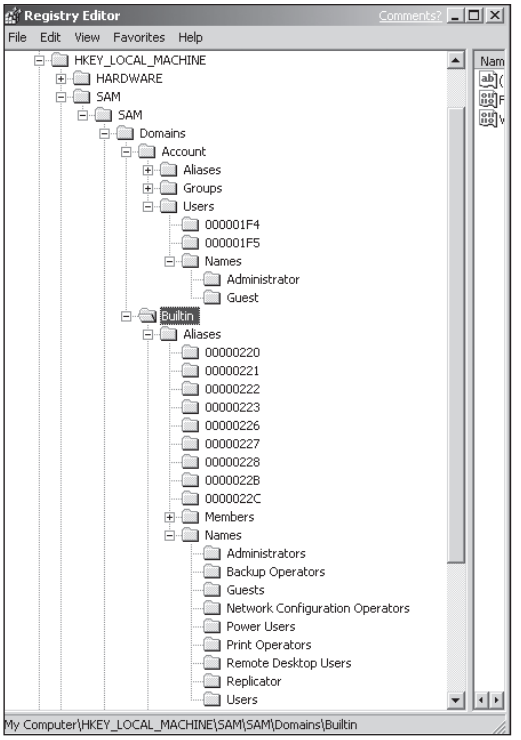

Figure 6.1 SAM database viewed by the Registry Editor after changing security permissions.

The total number of users, computers, and groups in classic NT is limited because the SAM cannot grow above a certain size. This is due to restrictions on overall Registry size called the *Registry size limit* (RSL). The RSL permits the Registry to grow to a maximum of 80 percent of paged pool memory. Paged pool memory has a ceiling of 192MB in NT and 470MB in Windows 2000 and Windows Server 2003.

Memory Pool Registry Settings

Memory used by the kernel in all Windows server products is divided between non-paged pool memory and paged pool memory. You can view settings for the memory pools in the following Registry key:

```
HKLM | System | CurrentControlSet | Control | Session Manager | Memory Management
```

The default values are zero, indicating that the system calculates them dynamically. You should not change any values without specific direction from Microsoft Product Support Services.

In modern Windows, the RSL is adjusted automatically when the Registry is about to exceed the current RSL setting. The RSL can also be adjusted from the User Interface (UI) using the Computer Management console as follows:

1. Launch the Computer Management console by entering COMPMGMT.MSC from the Run window.
2. Right-click the Computer Management icon and select PROPERTIES from the flyout menu.
3. Select the Advanced tab.
4. Click Performance Options.
5. Under Virtual Memory, click Change. This opens the Virtual Memory window.
6. Put a new value in the Maximum Registry Size field.

The SAM is only one component of the Registry, so its size is restricted still further. A classic SAM has enough room for about 40,000 users if you count the groups you'll need to manage them. Practical limits on replication and user management reduces this number considerably, although I know of at least one company that has in excess of 60,000 users in a single classic NT domain.

Single Point of Failure

The PDC is the only server that has read/write access to the SAM in a classic NT domain. If the PDC crashes or the telecommunications link to it goes down, you cannot make any changes to the domain constituents. You cannot add new users to a group. You cannot join computers to the domain. Users can still log on via a backup domain controller (BDC) but they cannot change their passwords.

To correct this problem, an administrator must promote a BDC to PDC somewhere in the domain. If the promoted BDC doesn't have the horsepower of the original PDC, worldwide performance suffers. A worse situation occurs if the WAN connection that connects the PDC to the rest of the domain goes down. In this situation, you don't dare promote a BDC because when the WAN connection returns, you'll have two PDCs with slightly different security database contents. This forces you to make a Solomon-like decision to keep one PDC and kill the other. In short, you have the makings of a real disaster.

Poor Operational Performance

The single PDC in a classic NT domain also imposes practical limits on daily operations. Assume, for example, that you are an administrator of a global NT network with 30,000 users. You are stationed in Omaha but the PDC for the master security

domain is in Boston. You open User Manager for Domains to add a new user. User Manager pulls the account database from the SAM on the PDC, not a local BDC. Depending on the speed of the intervening WAN links, it can take a long, long time to scan through a big SAM. Administrators in large NT domains learn to use command-line utilities to avoid this irritation.

Poor Replication Performance

The hub-and-spoke replication model of classic NT imposes operational limits beyond the problem with limited SAM size. A large network with many BDCs imposes a great deal of load on the PDC to keep the databases replicated. By default, replication occurs when 200 updates accumulate every seven minutes or at a random interval between one and seven minutes. If you don't want to wait for replication to carry an update to a remote BDC, you must use Server Manager to force replication. This means opening still another tool and waiting another period of time.

SAM Database Differs Between Servers and Domain Controllers

The SAM database has a different structure on a classic domain controller than on a regular server. For this reason, a classic NT server cannot be promoted directly to domain controller or demoted from a domain controller down to a server. You must reinstall the operating system completely to change the server's security role.

Lack of Management Granularity

A major weakness in the flat-file SAM structure is its inability to support hierarchical management. Administrators wield supreme power in a domain. A few BuiltIn groups such as Account Admins and Server Operators have specially tailored privileges, but there is no provision for localizing admin rights or creating new groups with a different set of limited rights. Third-party tools are available to overcome this lack of management granularity, but they carry their own replication and management baggage along with a hefty price tag.

Nontransitive Trust Relationships

Of all the limitations in classic NT, the ugliest is the inability to link domains together seamlessly while maintaining separate administrative roles.

Classic domains are linked by *trust relationships*. Domain controllers in trust-*ing* domains perform *pass-through authentications* to check the credentials of users from trust-*ed* domains. These trust relationships are based on entries in the NT Security database called *LSA Secrets*. (LSA stands for Local Security Authority.) A pair of LSA Secrets, one in each SAM database, links the two domains together.

Classic trust relationships can only operate in one direction. You can add complementary pairs of trusts to get the appearance of two-way authentication, but the two trusts operate independently.

Worse yet, classic trusts cannot extend beyond the two domains that form the trust endpoints. For instance, if Domain A trusts Domain B and Domain B trusts Domain C, then Domain A does not trust Domain C, or vice versa. This forces large NT systems to have many interlocking trusts. You know when you walk into the operations center of a big NT shop because there's butcher paper on the walls with circles and arrows going everywhere.

Multiple Logon IDs

In an ideal universe, a single network logon account would provide access to all server-based applications. In the past, application designers have been reluctant to base their authentication services on the classic NT logon mechanism. Part of this reluctance was due to the inscrutable set of security APIs that Microsoft provided. Designers were also put off by the inflexible nature of the SAM.

This means trying to achieve true single sign-on under NT has been very difficult. This forces users to memorize passwords for many different applications as well as their network logon. Because users often select the same password for different applications, the entire security system becomes as secure as the most vulnerable interface.

Improvements Made by Active Directory

Now that I've listed the litany of sins in classic NT, let's take a quick look at what Active Directory does to resolve them:

- The Active Directory account database in Windows Server 2003 can hold a billion objects. This resolves scalability concerns.
- Multiple domain controllers can host read/write copies of Active Directory, eliminating the problems with a single point of failure and poor operational performance.
- The Active Directory replication engine can be tuned to make best use of available bandwidth. This reduces WAN traffic.
- A modern Windows server (Windows 2000 or Windows Server 2003) can be promoted to a domain controller and demoted back to a member server without the need to reinstall the operating system.
- Active Directory can be configured with as many branches as needed to localize and compartmentalize administrative functions.
- Active Directory domains still use "trusts" as an operational model but the trusts now give full, two-way access to resources and are fully transitive between domains.
- The presence of a truly world-class directory service in Windows has sparked renewed interest among application developers in achieving single sign-on. Microsoft has helped encourage this interest by simplifying the security access methods and greatly expanding the access interfaces.

So, now we know what we're leaving behind. Let's move on to see what we're getting. The next section describes what goes into a directory service.

Directory Service Components

A directory service compiles information about objects of interest in the world and dispenses that information when given a suitably formulated request. The Yellow Pages are a kind of directory service. A library card catalog is another.

People like to have their information classified for easy retrieval. For instance, the Yellow Pages has categories like "Theaters—Movies" and "Restaurants—Outrageously Overpriced." A library card catalog classifies items into "Books—Fiction," "Books—Nonfiction," "Periodicals," and so forth.

Information needs to be readily accessible, as well. People want one-stop shopping. At the same time, you don't want all the information at a single location. This produces bottlenecks, single points of failure, and turf hassles. For this reason, the information in a directory service needs to be distributed among many sources. Each source of information is responsible for maintaining its little piece of the distributed database.

Information needs to follow rules to make it consistent and reliable. Yellow Pages ads contain a limited set of information about businesses in a community. You would not go to the Yellow Pages to look up the current stock price of a company.

A network directory service has entries for users and groups, workstations and servers, policies and scripts, printers and queues, switches and routers, and just about anything else that relates to computing. The attributes for these entries have something to do with their relationship to network services. For example, authentication credentials can be stored in a directory service so users can log on from anywhere the directory service is available. On the other hand, you would not expect to see a user's cologne preference in the directory service.

A directory service is not a general-purpose database. You would not implement a directory service to manage a point-of-sale system in a chain of video stores. But you would consider implementing a directory service to manage the salespeople who log on at the point-of-sale terminals.

Finally, a directory service needs management tools. Administrators need some way to add information to the directory, remove outdated information, and make use of the information that remains. These tools need to be global in scope, straightforward to operate, and aid in diagnosing any problems that might arise.

So let's get down to some basic questions. How does a directory service work? Why does it work that way? How does it break? How is it fixed? And most important, how does it make my job easier so I don't spend all my spare time managing the service that's supposed to be helping me manage the network?

Brief History of Directory Services

There's an old saying that you can't get to where you're going unless you know where you've been. Before analyzing Active Directory, let's start with a look at the history of directory services in general. This is not an academic exercise. It's important to understand the reason behind the decisions made when directory services were formulated and who made those decisions.

ITU-T

The directory service story starts with a smallish document called X.500, "Data Networks and Open System Communications—Directory." The cast of characters in this story includes a group of standards bodies and vendors from all over the world.

First and foremost is the *International Telecommunication Union* (ITU). The ITU is a United Nations agency that acts as a forum for governments that want to achieve consensus on global telecom issues. The ITU membership includes manufacturers and service providers from over 130 countries.

The branch of the ITU specifically tasked with making directory service recommendations is the Telecommunication Standardization Sector, or ITU-T. The ITU-T was formerly called the *Comité Consultatif International Téléphonique et Télégraphique* (CCITT).

The ITU-T issues recommendations in many areas, from broadcast requirements and measuring equipment to faxing. These recommendations are grouped into lettered series. For example, the V series covers data communication over telephone networks and includes such famous standards such as V.34, "Wideband Analog Modem Communication," and V.90, "Connecting Analog to Digital Modems."

The X series of recommendations, which includes the X.500 recommendations for directory services, covers a variety of data network and open system communication technologies, such as X.25 packet-switched networks and X.400 messaging systems. For a complete listing of ITU recommendations, see `www.itu.int/publications/telecom.htm`.

ISO

The ITU-T does not set standards; it only makes recommendations. Getting an international standard approved requires the consent of the *International Organization for Standardization* (ISO).

Source of the ISO Name

You may wonder why the initials ISO do not match the name, International Organization for Standardization. Actually, the letters are not initials at all. They come from the Greek word *isos*, meaning equal. These letters were used to avoid the hodgepodge of acronyms that would have resulted if the various member countries translated International Organization for Standardization into their own language with their own initials.

Unlike the ITU, whose membership comes from industry vendors, ISO members come from national standards bodies. The U.S. member is the *American National Standards Institute* (ANSI). The ISO web site is located at `www.iso.ch`. The `ch` indicates that the site is in Switzerland, just in case you are not up on your ISO 3166 two-letter country codes.

The ISO is responsible for standardization in just about every area, from the quality standards of ISO 9000 to the standard paper sizes of ISO 216. In the networking industry, it is most famous for ISO 7498, "Information Technology—Open System Interconnection—Basic Reference Model," better known as the OSI Network Model.

ISO standards that affect data communication technology are often jointly published with the ITU-T. For example, the ISO standard that parallels the ITU-T X.500 recommendations for directory services is ISO 9594, "Information Technology—Open Systems Interconnection—The Directory." Because the ISO issues standards and the ITU-T issues recommendations, it is actually a misnomer to refer to the *X.500 Standard*, but this is commonly done because the two documents are identical.

IEC

The ISO is the senior standards body in the world, but it certainly is not the only one. Many agencies dip their spoons in the standards soup bowl and they sometimes slosh on each other. In the data communications field, there is overlap between standards published by the ISO and standards published by the *International Electrotechnical Commission* (IEC).

The IEC deals with international standardization for electronics, magnetics, electromagnetics, electroacoustics, telecommunication, and energy production/distribution. They promulgate terminology, symbols, measurement standards, performance standards, dependability, design, development, safety, and environmental standards. The U.S. member of the IEC is also ANSI. The ISO and IEC joined with the ITU in publishing the directory service standards. The IEC web site is located at `www.iec.ch`.

ANSI

In the United States, there is one senior standards body, ANSI. You are probably most familiar with ANSI for its work to standardize character-based data formats, although there are ANSI standards for just about anything. I used to work in the nuclear industry, where even the ballpoint pens were built to conform to an ANSI standard. The ANSI web site is `www.ansi.org`.

IETF

In a country where millions of people call television talk shows to give advice to total strangers about their sex lives, it should come as no surprise that many advisory bodies are eager to give input to ANSI. An advisory body with a great deal of influence

over implementation of the X.500 standard is the *Internet Engineering Task Force* (IETF). Its web site is located at `www.ietf.org`.

The IETF is an amalgam of vendors, developers, researchers, designers, and architects of all stripes who have an interest in the workings of the Internet. Special working groups within the IETF ride herd on Internet workings in collaborative effort called the *Internet Standards Process*, a unique and somewhat lengthy operation that consists of thrashing a good idea mercilessly until it breaks into pieces that can be easily digested by the collective organism.

Request For Comments (RFC)

The Internet Standards Process is facilitated by documents called *Request for Comments* (RFCs) and *Internet Drafts*. To give you an idea of how long it takes to assimilate new ideas into Internet standards, out of the hundreds and hundreds of standards-track RFCs listed in RFC 2700, "Internet Official Protocol Standards," there are only 59 standards. The rest of the documents squirm somewhere in the approval process.

Copies of RFCs, Standards, Standards Track documents, Internet Drafts, and other working papers can be found at the IETF site and at various mirrored sites around the Internet. I prefer the search engine at the Internet Engineering Standards Repository, `www.normos.org`.

The IETF can bypass ISO/IEC standards and ITU recommendations if they deem it necessary to get useful protocols out into the world. An example of this is the *Lightweight Directory Access Protocol* (LDAP). LDAP is a pared-down version of the X.500 directory service that forms the basis of Active Directory, Netscape Directory Services, and other products.

There is no LDAP standard from ISO and no LDAP recommendation from the ITU. LDAP is purely an Internet concoction. Active Directory implements the most current version of LDAP, version 3, as documented in RFC 2251, "Lightweight Directory Access Protocol v3." This RFC expands and augments the original LDAP Standards Track document, RFC 1777, "Lightweight Directory Access Protocol." There is a long list of RFCs that expand various LDAP features.

Although LDAP is not precisely an X.500 implementation, a great deal of the design basis of LDAP comes from X.500. So before going through LDAP in detail, let's take a quick look at its parent.

X.500 Overview

A directory service is a distributed store of information about the users of a computer system and the infrastructure that supports that system.

The goal of X.500 was to cut through the babble of competing information repositories to define a single place where users from all nations could go to locate each

other, learn about each other, discover common likes and dislikes, and eventually communicate freely to find a path to universal peace and brotherhood and the dawning of the Age of Aquarius. The key features of an X.500 directory service are as follows:

- The information is distributed among many different servers.
- Users can submit queries to any server to find information anywhere in the system.
- Servers can find information on other servers because they share common knowledge about each other.

X.500 Components

The magic of X.500 comes from the flexible way it compartmentalizes and distributes information. This flexibility comes at the cost of complexity, though—not the least of which is a thicket of nomenclature rife with obscure computing jargon and Three Letter Acronyms (TLAs). These X.500 acronyms crop up quite a bit in Active Directory documentation, so it pays to give them a Quick Run Through (QRT). Refer to Figure 6.2 for a roadmap. Here are the X/500 TLAs:

- Information in an X.500 Directory is stored in a *Directory Information Base* (DIB).
- The DIB is divided into pieces that are structured into a hierarchy called a *Directory Information Tree* (DIT).

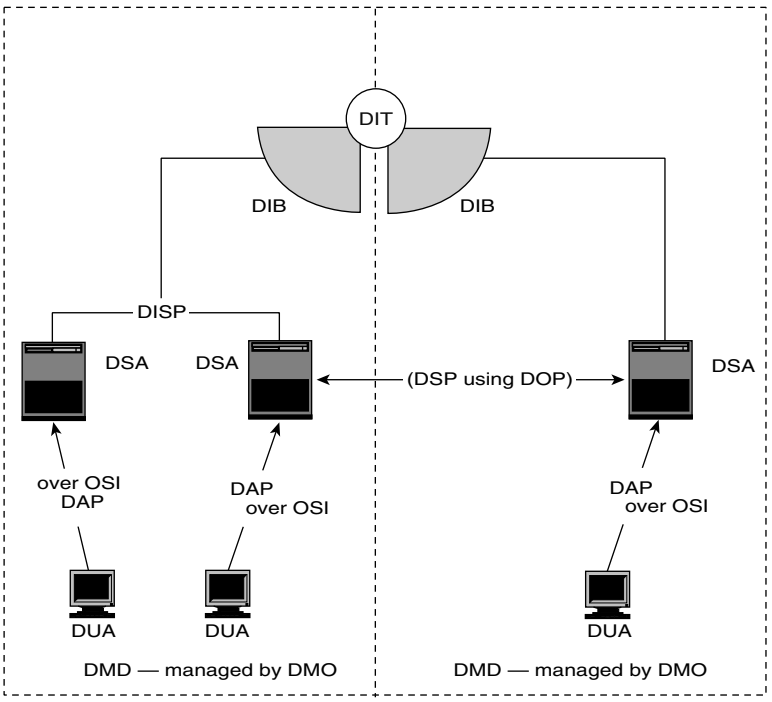

Figure 6.2 X.500 components and their communication protocols.

- Each piece of the DIB is stored on a server called a *Directory Service Agent* (DSA).
- A user who needs information from Active Directory submits queries via an application interface called a *Directory User Agent* (DUA).
- A DUA communicates with a DSA using the *Directory Access Protocol* (DAP).
- One DSA communicates with another using the *Directory System Protocol* (DSP).
- Administrative information exchanged between DSAs is controlled via policies defined by the *Directory Operational Binding Management Protocol* (DOP).
- A single *Directory Management Organization* (DMO) takes charge of a *Directory Management Domain* (DMD) that contains one or more DSAs.
- Information held by one DSA is replicated to other DSAs in the same DMD using the *Directory Information Shadowing Protocol* (DISP).

DAP, DSP, DISP, and all other high-level communication protocols in X.500 use OSI networking as defined in ITU Recommendation X.200/OSI-EIU Standard 7498.

X.500 Transaction Example

Here's an example of how these X.500 components tie together (see Figure 6.3). Let's say that the secondhand car dealers in America get together and decide to form an association. They want a directory service to store information about vehicles available for sale at each member's showroom.

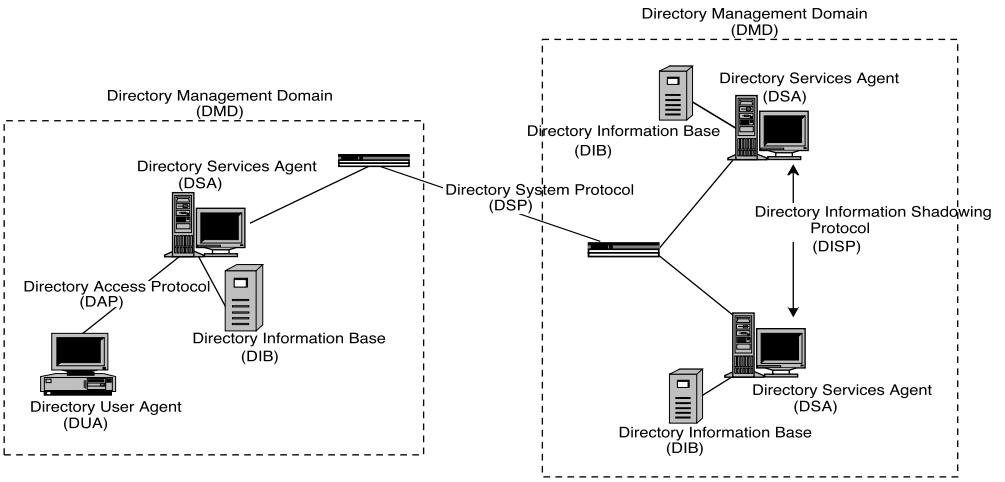

Figure 6.3 Diagram of an example X.500 communication scheme.

The DIB for this dealership directory service includes makes, models, years, vehicle identification numbers, and unbeatable prices. Each dealer is assigned a DMO that controls a DMD. The DIB in each DMD is hosted by at least one DSA, which exchanges administrative information with DSAs in other DMDs using DOP. Dealerships in the same region have individual DSAs that replicate their copy of the DIB between each other via DISP. The pieces of the DIB are joined into a single DIT, the root of which is hosted by a DSA at headquarters.

Why go through all this trouble? Well, if a customer at a dealership in Kankakee wants a cherry-colored Cherokee, the salesperson can sit at a DUA and submit a query to a local DSA via DAP. The DSA would check its copy of the local DIB and if it failed to locate a record, it would use DSP to query other DSAs until it either found a match or exhausted all possibilities. The DUA could then be programmed to suggest alternatives, like a cream-colored Chevelle in Chicago.

The important point to remember about this transaction is that there is no central repository of information. Each local DSA holds its own copy of the DIB. Referral mechanisms are used to distribute queries around the system.

Why LDAP Instead of X.500?

Several pedigreed X.500 directory services are commercially available, but few have achieved widespread popularity. The problem with pristine X.500 implementations is the overhead represented by all those protocols. When you get an army of DUAs all talking DAP to DSAs that refer queries to other DSAs using DSP while at the same time mirroring their DIBs to other DSAs in their DMD via DISP, my friend, you've got a whole D* lot to go wrong.

In the early 90s, a few bright folks at the University of Michigan wanted to build a directory service to handle their 100,000+ students, staff, and faculty. They gave up on the complexities of X.500 and came up with a scheme that retained the X.500 directory structure but gave it a streamlined access protocol based on standard TCP/IP instead of ISO. They also came up with a pared-down referral mechanism, a more flexible security model, and no fixed replication protocol. They called the result the *Lightweight Directory Access Protocol*, or LDAP. The rest, as they say, is history. The Blue and Maize folks no longer control LDAP development. The current repository of LDAP knowledge is at `www.openldap.org`.

Active Directory and LDAP

When Microsoft decided to replace the clumsy Registry-based account management system in classic NT with a true directory service, rather than devise a proprietary directory service of their own, they chose to adopt LDAP. Even more importantly, from our perspective as administrators, Microsoft chose to deliver their LDAP directory service using two proven technologies.

Extensible Storage Engine (ESE)

At its heart, a directory service database is made up of tables with rows representing objects of interest and columns representing attributes of those objects. What sets different databases apart is the way the tables are managed. This table manager is often called a *database engine.*

The LDAP standards do not stipulate a particular table management technology. For the Active Directory table manager, Microsoft used a revved-up version of the *Extensible Storage Engine* (ESE) first introduced with Exchange. Microsoft chose ESE over the SQL Server database engine because a SQL engine does not work efficiently with the object-oriented structure of an LDAP directory. The ESE engine, on the other hand, was primarily designed as an object-oriented database.

DNS-Based Locator System

Users cannot take advantage of the information in a directory service if they cannot find the servers hosting the information. Microsoft chose to build its LDAP directory service around the *Domain Name System* (DNS). When an LDAP client needs to find a server hosting a directory service, it does so by querying DNS. This enabled Microsoft to use new features in DNS to simplify the search.

For example, Microsoft took advantage of the relatively new service locator (`SRV`) record type to put pointers in DNS to indicate the names of servers hosting LDAP and Kerberos services. SRV records have a relatively complex structure, but Microsoft was able to avoid typographical errors by registering them automatically using Dynamic DNS.

LDAP Information Model

A directory service may be a bit fancier than the database you use to tally the overtime pay you've lost since taking your salaried administrator position a few years back, but the principles of operation are pretty much the same.

Object-Oriented Database

In X.500 terminology, the directory service database is called a *Directory Information Base* (DIB). If you think of an old-style library card catalog system as a kind of directory service, one of those big oak cabinets with rows of drawers would be a DIB.

The X.500 directory service structure was developed at a time when object-oriented databases represented leading-edge technology. If your only exposure to database technology has been more modern relational databases, the design constraints of an object database can look a little strange.

In an object-oriented database, each record (object) occupies a unique position in a hierarchical namespace. The object's name and path traces its origins to the top of

the namespace, in much the same way that a Daughter of the American Revolution traces her forebears back to the Mayflower. A file system is an example of an object-oriented database.

Object databases consist of big, structured sequential files connected by a set of indexes that are themselves nothing more than big, structured sequential files. This underlying database technology is called *Indexed Sequential Access Method*, or ISAM. You'll see this term in the Event log and other reports.

The ESE database engine exposes the flat ISAM structure as a hierarchy of objects. In addition, Microsoft makes extensive use of COM technology by representing Active Directory objects as COM objects via the *Active Directory Services Interface* (ADSI).

Classes and Attributes

A directory service contains information about specific types of objects, such as User objects, Computer objects, and so forth. These are called object *classes*. A class is a bundle of *attributes* with a name. Figure 6.4 shows how attributes and classes are related.

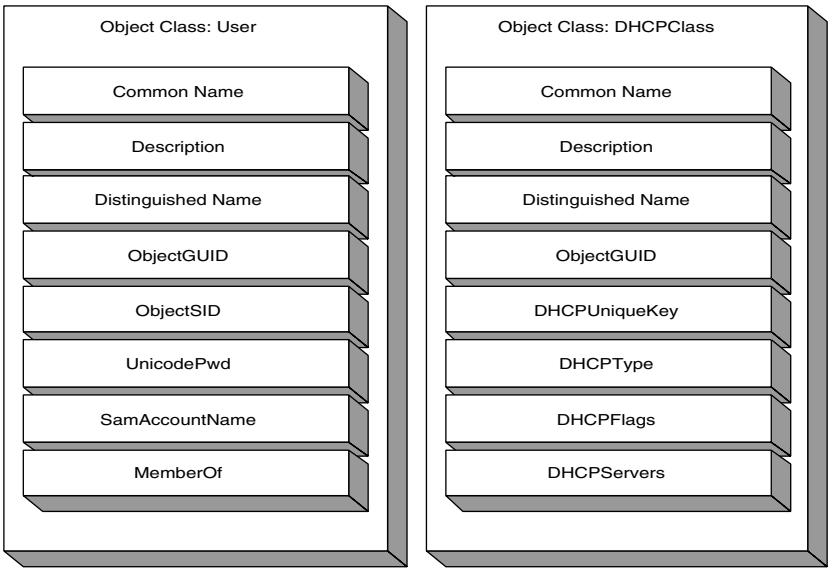

Figure 6.4 Classes and attributes in a directory service.

Attributes and Properties

Attributes are also often called *properties*. There is a difference between these two terms, but it is so subtle that most reference manuals, including this one, use them interchangeably.

The attributes associated with a particular object class differentiate it from other object classes. For example, *User* objects have different attributes than *Computer* objects or *IP Security* objects. Using a library card catalog as an example, different card formats represent different classes of items. A certain card format is used to record entries for `Books`. Another format is used for `Tapes`. The card format for `Books` would have spaces for `Title`, `Author`, `ISBN`, and so forth. A card for `Tapes` would have spaces for those entries plus additional spaces for `Read-By` and `Play-Time`.

An object class, then, is really nothing more than a bundle of attributes with a name. RFC 2256, "A Summary of the X.500(96) User Schema for use with LDAPv3," defines 21 classes and 55 attributes for use in a standard LDAP directory service. Active Directory adds quite a few more for a total of about 200 object classes and 1500 attributes.

Classes also define the scope of a directory service database. You would not expect to find cards in a library card catalog representing Off-The-Road Vehicles or Double-Meat Hamburgers. Microsoft engineers defined the initial scope of Active Directory by including a certain set of object classes and attributes. This list can be extended by other applications or by administrators. For example, your organization could create attributes and classes for storing badge numbers and social security numbers in Active Directory.

Class Inheritance

Directory service designers strive to limit complexity by defining the minimum number of classes and attributes necessary to describe the objects of interest that need to be stored in the directory service database.

For example, in a library card catalog, it would be a mistake to create a class called Somewhat-Less-Than-Riveting-Early-20th-Century-American-Novels, even though it seems like quite a few objects would fit that class. In relation to the overall scope of a library, this classification would be too narrow. It would be better to have an attribute called Boring with a Boolean value. You could assign this attribute to the Book class so that objects derived from that class would get a Boring attribute that could be given a value of Yes or No or left empty. You could also assign the Boring attribute to the Periodical, Tape, and Video classes, as well.

A directory can have hundreds of classes and many hundreds of attributes. If the attributes for each class had to be separately defined, the sheer number of perturbations would make the directory look less like a tree and more like an example of German expressionism.

Fortunately, attributes associated with a particular class often overlap those of other classes. For example, the attribute list for the Mailbox class includes all the attributes associated with the Mail-Recipient class with one addition, the Delivery-Mechanism attribute. So, instead of separately defining all the attributes in Mailbox class, LDAP allows the class to be defined as a child of the Mail-Recipient class. This permits it to

inherit the attributes of its parent. The designer need only stipulate the new additional attribute or attributes that make the subordinate class unique.

Attributes flow down the hierarchy of object classes like genes in a family tree. Figure 6.5 shows an example of class inheritance for the Computer object class.

All LDAP classes derive from a class called *Top*. This makes it possible to define certain attributes that every class would have in common. For example, every class needs a *Common-Name* attribute. The attribute is assigned to Top and the rest of the classes inherit it.

Think of Top as a director who never actually appears on camera but leaves a distinctive mark on the production. Top is an Abstract class, one of three class types in LDAP. They are as follows:

- **Abstract.** Classes that exist solely to derive other object classes. There are 14 abstract classes in the Active Directory. Examples include Top, Device, Person, and Security Object.
- **Structural.** Classes that have objects in Active Directory. Examples include User, Group, and Computer.
- **Auxiliary.** Used to extend the definition of an Abstract class for specialized purposes. There are only six of these classes in Active Directory: Mail-Recipient, Dynamic-Object, MS-MMS Object, Sam-Domain, Sam-Domain-Base, and Security-Principal.

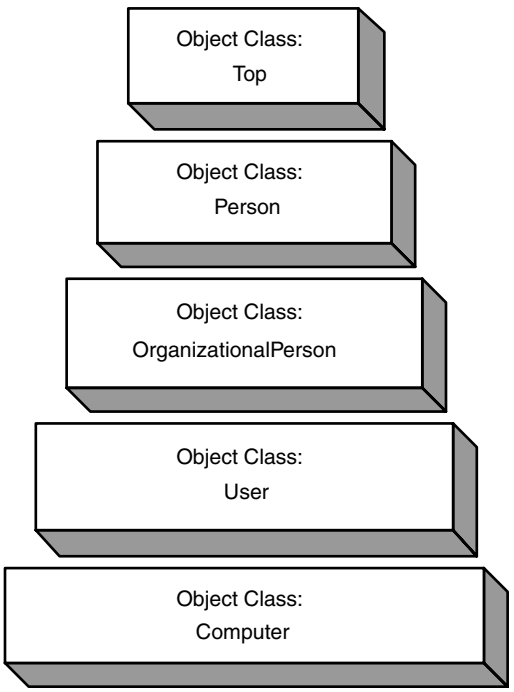

Figure 6.5 Inheritance diagram for the Computer object class.

These three class types act like assembly line robots designed to produce things called "objects." The Structural classes are the tools and dies that stamp and shape the objects. The Abstract classes are the mill workers and pattern makers that build the tools and dies. The Auxiliary classes act like a custom shop at the end of the line where special versions of standard objects are turned out.

Object Instances

Each object in Active Directory is derived from a specific object class. Another way of saying this is that an object represents an *instance* of a class. Each instance of an object class differs from another instance by having different values for its attributes.

Remember the movie *Elephant Man*? In a great scene, the lead character, John Merrick, stands in front of a curious mob and exclaims, "I am not an elephant. I am a human being." Had Mr. Merrick been a directory services designer, he could have clarified his point by adding, "I am an instance of the Human Being class, not the Elephant class. And the only difference between you and me is a relatively minor attribute of mine that has a different value from yours. So lay off, will you?"

Defining suitable attributes for an object class can be slippery. Subtle differences may force a designer to create a new class. If you were designing a library card catalog, you might start out by defining a class called Tape with an attribute called Type that has two permitted values, Audio and Video. This decision forces you to define attributes for the Tape class that fully defines both audiotapes and videotapes. After months of agonizing, you might decide that the properties of audio and video tapes are so different that they warrant creating two classes, AudioTape and VideoTape, each with their own unique attribute sets. There are many instances in Active Directory and LDAP where two object classes differ by only one or two attributes.

Schema

A database schema defines the content and structure of the database. In a library card catalog, the schema would be a set of procedures and rules set down by the librarian. "Books go on green cards," she tells you. "Videos go on red cards. File the cards alphabetically by Title in this cabinet and by Subject in that cabinet." So on and so on. The schema for an LDAP directory service defines these items:

- The attributes associated with each object class
- The permissible object classes
- The parent-child relationship of object classes, which in turn determines attribute inheritance
- The data type associated with each attribute
- The physical representation of the object in the user interface

The schema can take the form of an external table that acts as data dictionary or an internal table that is structured using the same rules as the database itself. Active Directory uses an internal schema. Many of the design constraints we'll see in the next chapter stem from the necessity to keep a consistent schema throughout all the servers that host a copy of the directory database.

Later in this chapter, we'll see how to modify the Active Directory schema to add new attributes and object classes that can be used by applications to support network operations.

LDAP Information Model Summary

Here are the important information model concepts to carry forward with you when you start designing an Active Directory system for your own organization:

- LDAP uses an object-oriented database. The database engine for Active Directory is the Extensible Storage Engine, or ESE.
- An object class defines a unique set of attributes for a particular type of object.
- Object classes inherit attributes from their parents. This permits the designer to identify only the new attributes for a new object class.
- Each object is an instance of an object class. The attributes for the object are assigned values that describe that particular object.
- A schema defines the content and structure of the LDAP database. In the case of Active Directory, the schema is contained within the directory itself.
- The directory schema must be consistent on every server hosting a copy of the database.

LDAP Namespace Structure

A directory service has two major features. First, it distributes its information base among many different servers. Second, users can access directory information by querying any of those servers. Making this work requires defining a *namespace* in which each object's location can be quickly determined.

Common Names

As we saw in the last section, information in an LDAP database comes in the form of objects. Objects have attributes that describe them. For example, the User object for Tom Jones would have attributes such as Tom's logon name, his password, his phone number, his email address, his department, and so forth.

When an LDAP client needs to locate information about an object, it submits a query that contains the object's distinguished name (DN) and the attributes the client wants to see. A search for information about Tom Jones could be phrased in a couple of ways:

- You could search for attributes in Tom's User object. "Give me the Department attribute for `cn=Tom Jones,cn=Users,dc=Company,dc=com`."
- You could search for attributes that end up including Tom's object. "Give me all User objects with a Department attribute equal to `Finance`."

In either case, LDAP can find Tom's object because the name assigned to the object describes its place in the LDAP namespace.

Figure 6.6 shows a portion of the LDAP namespace in Active Directory. With one exception, each folder represents a Container object, which in turn holds other objects. The exception is the domain controllers object, which is an Organizational Unit (OU). Domain controllers are placed in an OU so that they can have discrete group policies. Generic Container objects cannot be linked to group policies.

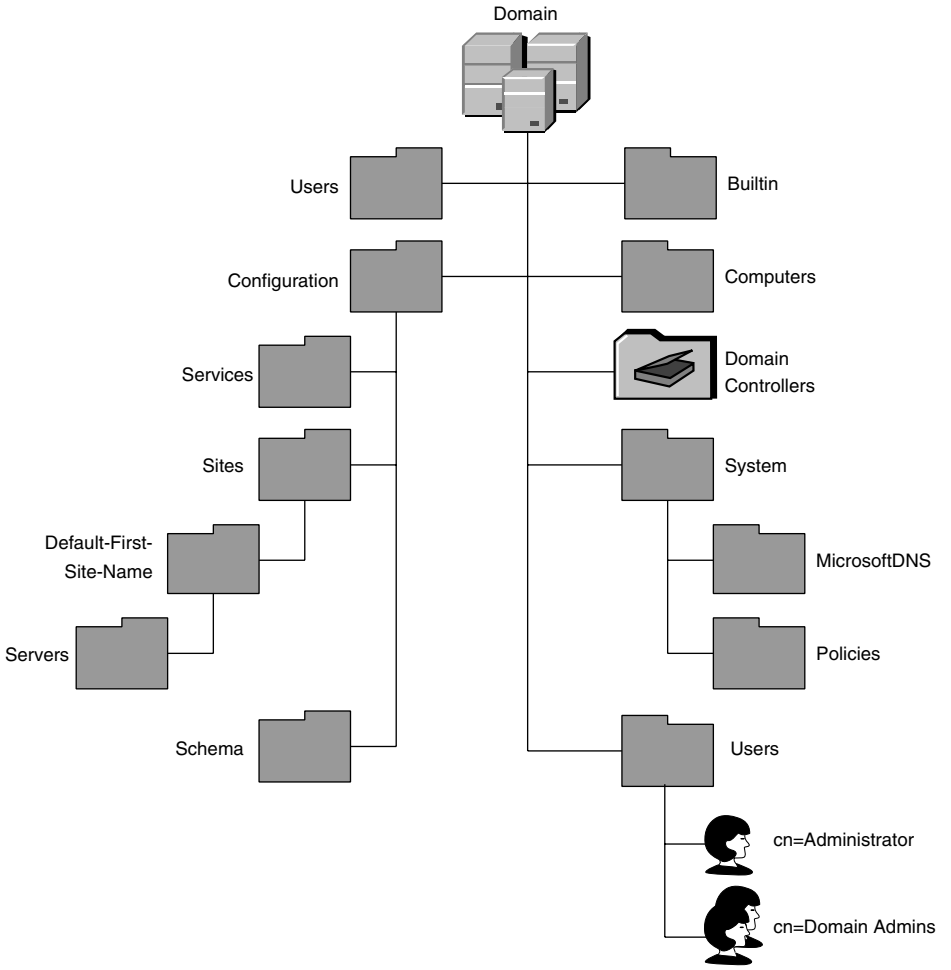

Figure 6.6 Example LDAP directory hierarchy.

The User objects in the diagram have designators that start with CN, meaning *Common Name*. The CN designator applies to all but a few object types. Active Directory only uses two other object designators (although LDAP defines several). They are as follows:

- **Domain Component (DC).** DC objects represent the top of an LDAP tree that uses DNS to define its namespace. Active Directory is an example of such an LDAP tree. The designator for an Active Directory domain with the DNS name Company.com would be `dc=Company,dc=com`.
- **Organizational Unit (OU).** OU objects act as containers that hold other objects. They provide structure to the LDAP namespace. OUs are the only general-purpose container available to administrators in Active Directory. An example OU name would be `ou=Accounting`.

Distinguished Names

A name that includes an object's entire path to the root of the LDAP namespace is called its *distinguished name*, or DN. An example DN for a user named CSantana whose object is stored in the `cn=Users` container in a domain named `Company.com` would be `cn=CSantana,cn=Users,dc=Company,dc=com`.

An identifying characteristic of LDAP distinguished names is their little-endian path syntax. As you read from left to right, you travel up the directory tree. This contrasts to file system paths, which run down the tree as you read from left to right.

Relative Distinguished Names

An object name without a path, or a partial path, is called a *relative distinguished name*, or RDN. The common name `cn=CSantana` is an example of an RDN. So is `cn=CSantana,cn=Users`. The RDN serves the same purpose as a path fragment in a filename. It is a convenient navigational shortcut.

Two objects can have the same RDN, but LDAP has a rule that no two objects can have the same DN. This makes sense if you think of the object-oriented nature of the database. Two objects with the same DN would try to occupy the same row in the database table. *C'est impossible*, as we say in southern New Mexico.

Case Sensitivity of LDAP Names

Distinguished names in Active Directory are not case sensitive. In most instances, the case you specify when you enter a value is retained in the object's attribute. This is similar to the way Windows treats filenames. Feel free to mix cases based on your corporate standards or personal aesthetic.

Typeful Names

The combination of an object's name and its LDAP designator is called a *typeful* name. Examples include `cn=Administrator` and `cn=Administrator,cn=Users,dc=Company, dc=com`.

Some applications can parse for delimiters such as periods or semicolons between the elements of a distinguished name. For example, an application may permit you to enter `Administrator.Users.Company.com` rather than the full typeful name. This is called *typeless* naming. When entering typeless names, it is important to place the delimiters properly.

The console-based tools provided by Microsoft use a GUI to navigate the LDAP namespace, so you don't need to worry about interpreting typeful or typeless names right away. But if you want to use many of the support tools that come on the Windows Server 2003 CD or in the Resource Kit, or you want to use scripts to manage Active Directory, you'll need to use typeful naming. After you get the hang of it, rattling off a long typeful name becomes second nature.

Directory Information Tree

In LDAP, as in X.500, the servers that host copies of the information base are called *Directory Service Agents*, or DSAs. A DSA can host all or part of the information base. The portions of the information base form a hierarchy called a *Directory Information Tree*, or DIT. Figure 6.7 shows an example.

The top of the DIT is occupied by a single object. The class of this object is not defined by the LDAP specification. In Active Directory, the object must come from the object class *DomainDNS*. Because Active Directory uses DNS to structure its namespace, the DomainDNS object is given a DC designator. For example, the object at the top of the tree in Figure 6.7 would have the distinguished name `dc=Company,dc=com`.

Typeless Names and Delimiters

If you write scripts and you need to allow for periods in object names, precede the period with a backslash. This tells the parser that the period is a special character, not a delimiter. For example, if your user names look like `tom.collins`, a typeless name in a script would look like this: `tom\.collins.Users.Company.com`. The same is true for user names that have embedded commas and periods, such as `Winston H. Borntothepurple, Jr.` An ADSI query for this name would look like this: `winston h\. borntothepurple\, jr\.`

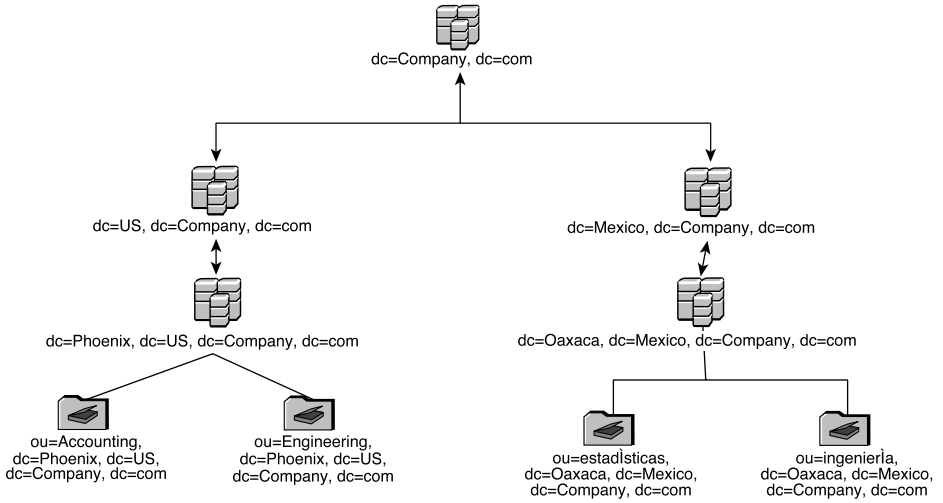

Figure 6.7 Directory Information Tree.

Active Directory and DNS Roots

Active Directory cannot be rooted at the very top of a DNS namespace. The assumption is that many different Active Directory namespaces could share the same root. For this reason, the DomainDNS object at the top of the tree must always have at least two domain component designators.

An LDAP tree contains branches formed by containers underneath the root container. These containers hold objects that have some relation to each other as defined by the namespace. For instance, in Active Directory, the default container for User objects is `cn=Users`. For Computer objects, it is `cn=Computers`. Information about group policies, DNS, Remote Access Services, and so forth go in `cn=System`. As we'll see when we discuss Active Directory design in Chapter 8, "Designing Windows Server 2003 Domains," administrators have the ability to create Organizational Units (OUs) to contain objects that have similar management or configuration requirements.

Naming Contexts

As the number of objects in a DIT grows, the database may get too large to store efficiently on one DSA. Also, an organization might want to use bandwidth more effectively by using a DSA in New York to store information about users in North America and another DSA in Amsterdam to store information about users in Europe.

Naming Contexts and Partitions

X.501, "Information Technology—Open Systems Interconnection—The Directory: Models," defines the term naming context as, "A subtree of entries held in a single master DSA." It goes on to describe the process of dividing a tree into multiple naming contexts as *partitioning*.

Novell chose to adopt the term *partition* to define separate pieces of the directory database. In their seminal book, *Understanding and Deploying LDAP Directory Services*, Tim Howe, Mark Smith, and Gordon Good use the term *partition* in favor of naming context, although they describe both as meaning the same thing. Microsoft uses the two terms interchangeably.

The tools that come with the Windows Server 2003 CD and in the Resource Kit favor the term *naming context*. That is the term I use throughout this book.

Here is where the distributed nature of an LDAP database comes into play. The Directory Information Base can be separated into parts called *naming contexts*, or NCs. In Active Directory, each domain represents a separate naming context. Domain controllers in the same domain each have a read/write replica of that Domain naming context. Configuration and Schema objects are stored in their own naming contexts, as are DNS Record objects when using Active Directory Integrated DNS zones.

When a client submits a query for information about a particular object, the system must determine which DSA hosts the naming context that contains that particular object. It does this using the object's distinguished name and knowledge about the directory topology.

If a DSA cannot respond to a query using information in the naming contexts it hosts, it sends the client a *referral* to a DSA hosting the next higher or lower naming context in the tree (depending on the distinguished name of the object in the search). The client then submits the request to a DSA hosting the naming context in the referral. This DSA either responds with the information being requested or a referral to another DSA. This is called *walking the tree*.

DSAs that host copies of the same naming context must replicate changes to each other. It's important to keep this in mind as you work with Active Directory servers. If you have separate domains, then clients in one domain must walk the tree to get access to Active Directory objects in another domain. If the domain controllers for the domains are in different locations in the WAN, this can slow performance. Many of the architectural decisions you'll make as you design your system focus on the location, accessibility, and reliability of naming contexts.

LDAP Searches

From a client's perspective, LDAP operates like a well-run department store. In a department store, you can sidle up to the fragrance counter and ask, "How much is the Chanel No. 5?" and be sure of getting an immediate reply, especially if you already have your credit card in hand. The same is true of LDAP. When a search request is submitted to a DSA that hosts a copy of the naming context containing the objects involved in the search, the DSA can answer the request immediately.

But in a department store, what if you ask the fragrance associate, "Where can I find a size 16 chambray shirt that looks like a Tommy Hilfiger design but doesn't cost so darn much?" The associate probably doesn't know, but gives you directions to the Menswear department. You make your way there and ask your question to an associate standing near the slacks. The associate may not know the answer, but gives you directions to the Bargain Menswear department in the basement behind last year's Christmas decorations. You proceed to that area and ask an associate your question again. This time you're either handed a shirt or given an excuse why one isn't available.

LDAP uses a similar system of referrals to point clients at the DSA that hosts the naming context containing the requested information. These referrals virtually guarantee the success of any lookup so long as the object exists inside the scope of the information base.

The key point to remember is that LDAP referrals put the burden of searching on the clients. This contrasts to X.500, where all the messy search work is handed over to the DSAs. LDAP is Wal-Mart to the Nordstroms of X.500.

RootDSE

When LDAP clients need information from a DSA, they must first *bind* to the directory service. This authenticates the client and establishes a session for the connection. The client then submits queries for objects and attributes within the directory. This means the client needs to know the security requirements of the DSA along with the structure of the directory service it hosts.

DSAs "advertise" this information by constructing a special object called *RootDSE*. The RootDSE object acts like a signpost at a rural intersection. It points the way to various important features in the directory service and gives useful information about the service. LDAP clients use this information to select an authentication mechanism and configure their searches.

Each DSA constructs its own copy of RootDSE. The information is not replicated between DSAs. RootDSE is like the eye above the pyramid on the back of a dollar bill. It sits apart from the structure but knows all about it. You'll be seeing more about RootDSE later in this book in topics that cover scripting. Querying RootDSE for information about Active Directory rather than hard-coding that information into your scripts is a convenient way to make your scripts portable.

LDAP Namespace Structure Summary

Here are the highlights of what you need to remember about the LDAP namespace structure to help you design and administer Active Directory:

- An object's full path in the LDAP namespace is called its distinguished name. All DNs must be unique.

- The Directory Information Tree, or DIT, is a distributed LDAP database that can be hosted by more than one server.
- The DIT is divided into separate units called naming contexts. A domain controller can host more than one naming context.
- Active Directory uses separate naming contexts to store information about domains in the same DIT.
- When LDAP clients search for an object, LDAP servers refer the clients to servers that host the naming context containing that object. They do this using shared knowledge about the system topology.
- Each DSA creates a RootDSE object that describes the content, controls, and security requirements of the directory service. Clients use this information to select an authentication method and to help formulate their search requests.

Active Directory Namespace Structure

At this point, we know enough about a generic LDAP directory service to begin applying the terms and concepts to Active Directory.

Let's start with what we need to store in Active Directory. You can classify the required information into three general categories:

- **Information about network security entities.** This includes users, computers, and groups along with applications such as group policies, DNS, RAS, COM and so forth.
- **Information about the Active Directory mechanisms.** This includes replication, network services, permissions, and user interface displays.
- **Information about the Active Directory schema.** This includes objects that define the classes and attributes in Active Directory.

Microsoft had to devise a way to structure this information in a way that was compatible with LDAP while retaining backward compatibility with classic NT. In classic NT, information about security entities is stored in the SAM and SECURITY databases in the Registry. Microsoft calls the contents of the SAM database a domain. Because the only way to control access to the SAM is to control access to the entries in the SAM, a domain defines a security boundary as well as a management boundary.

The SAM databases in classic NT domains cannot be combined. To get a common security boundary, the domains must be knitted together using *trust relationships.* When one domain trusts another, members of the trust-*ed* domain can be used as security entities in the trust-*ing* domain. The underlying authentication mechanism, NT LanMan Challenge Response, supports this trust relationship by permitting pass-through authentication of users from trusted domains.

Domains

LDAP gave Microsoft the freedom to construct just about any namespace it chose. There is no naming restriction other than at the top of the namespace, where the distinguished name of the root object needs to correspond to a DNS domain name.

Janis Joplin had it right, though, that freedom's just another word for nothing left to lose, and Microsoft had a lot to lose if it designed Active Directory in such a way as to not be compatible with classic NT. For this reason, Microsoft chose to structure Active Directory around the classic concepts of domains and trust relationships.

In Active Directory, a *domain* defines a separate namespace, a separate security structure, a separate management structure, and a separate naming context. The Active Directory database is hosted on *domain controllers*. Users and computers are *members* of a domain. Group policies are contained within a particular domain, even if they impact users in other domains.

Active Directory Naming Contexts

Active Directory is capable of holding a billion objects. This is enough to hold account, computer, mailboxes, and group memberships for every person in the western hemisphere. A big Active Directory database is like an NBA center, though. He may be the key to winning, but only if he doesn't have to move too fast or too often.

The Active Directory database, Ntds.dit, can grow very quickly. The DIT for a domain with 150,000 objects could be well over 2GB depending on the number of groups and the length of the group membership. A DIT this size can be difficult to replicate and manage. Also, it does not make sense to replicate information about users in one continent to domain controllers on other continents unless those users regularly share information.

Domain Naming Contexts

LDAP permits breaking up a directory into separate naming contexts. Managing the interfaces between these naming contexts can get a little tricky, though. To get maximum performance, it is often necessary to generate local caches containing references to objects in other naming contexts. Riding herd on these external reference caches to make sure they reflect the most current information takes some doing.

Microsoft chose to avoid many of the complexities involved with naming contexts by eliminating the ability to create ad hoc naming contexts. As an Active Directory administrator, you have only two places where you can create a naming context (see Figure 6.8):

- At a domain boundary
- By creating a special Application naming context (a new feature in Windows Server 2003)

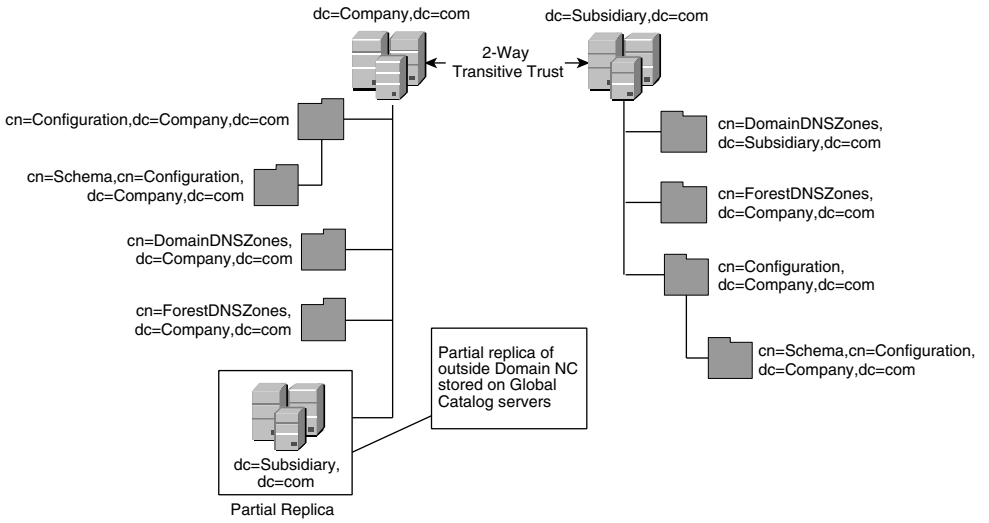

Figure 6.8 Active Directory forest showing naming contexts.

The Application naming context has only limited utility (it is currently used only to support DNS), so the only real option to break apart a big DIT is to create separate domains. In addition to the Domain naming context, each Active Directory implementation contains two other naming contexts: Configuration and Schema. Every domain controller in the forest gets a replica of these two naming contexts. The Schema replica is read-only except for the domain controller selected as the Schema Operations Master.

Schema Naming Contexts

The Schema naming context holds ClassSchema and AttributeSchema objects that represent the various classes and attributes in Active Directory. If this sounds like a circular definition, it's meant to be. Unlike some directory services that load the schema in a separate file, the Active Directory schema is completely self-referential.

Every domain controller in a forest hosts a read-only copy of the Schema naming context. Only one domain controller, the Schema Role Master, can make changes to the schema.

The Schema container object is an instance of the Directory Management Domain (DMD) class. This is a holdover from Exchange, which uses X.500 terminology to define the information store. Because the Schema object represents a naming context boundary, it also contains replication control attributes similar to those in the Configuration object and the Domain-DNS object.

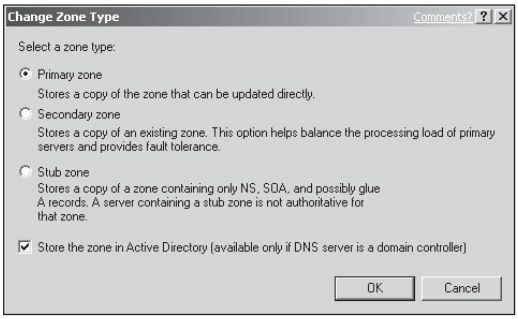

Figure 6.9 DNS zone properties—Change Zone Type window showing `Active Directory Integration` option.

If you search through the objects in the Schema container, you'll come across a special object called Aggregate. This lone instance of the LDAP SubSchema class has attributes called AttributeTypes and ObjectClasses that lists the names of all classes and attributes in Active Directory. LDAP clients query for the contents of this object to discover the structure of the directory. This helps them formulate queries.

Application Naming Contexts

A new feature in Windows Server 2003 is the ability to create additional naming contexts that can be placed on specific domain controllers. Microsoft uses this feature to store DNS resource records for Active Directory Integrated zones.

You elect to Active Directory Integrate a zone using the Properties of the zone in the DNS console. The `General` tab displays the zone type. Click `Change` to open the Change Zone Type window that lists your options. Figure 6.9 offers an example.

If you elect to integrate a zone into Active Directory, the resource records are copied from the existing text-based zone file into Active Directory as discrete `DNSzone` objects. In Windows 2000, these objects are stored in a `MicrosoftDNS` container in `cn=System,dc=<domain>,dc=<root>`. This gave limited flexibility to administrators who wanted to deploy Active Directory Integrated DNS in large, multidomain forests.

The application naming contexts added by Windows Server 2003 gives this additional flexibility. When you Active Directory Integrate a zone on a DNS domain controller running Windows Server 2003, the domain controller creates two additional naming contexts:

- **DomainDNSZones.** A replica of this naming context is placed on domain controllers running the DNS service. Each domain gets a separate DomainDNSZones NC.

- **ForestDNSZones.** A replica of this naming context is placed on domain controllers running DNS throughout the forest.

When you elect to Active Directory Integrate a zone, a new entry called `Replication` is added to the `General` tab in the zone Properties window. Click the `Change` button next to this entry to open the Change Zone Replication Scope window (see Figure 6.10).
This window gives you the following replication options:

- **`All DNS servers in the forest.`** If you select this option, the zone records are placed in the ForestDNSZones naming context. This is the broadest scope and involves the most replication traffic.
- **`All DNS servers in the domain.`** This option places the resource records in the DomainDNSZones naming context for the domain of the DNS server. For instance, if you create stub zone on a DNS server in `Company.com` that points at Branch.Company.com, the records in the stub zone would be placed in `cn=DomainDNSZones,dc=Company,dc=com`.
- **`All domain controllers in the domain.`** This option places the zone records in the Domain naming context under `cn=MicrosoftDNS,cn=System,dc=<domain>, dc=<root>`. This is the same container used by Windows 2000, so select this option when you have Windows 2000 DNS server hosting Active Directory Integrated zones.
- **`All domain controllers specified in the scope of the application directory partition.`** This option permits you to select a specific application naming context.

If you have a single domain, there is nothing to be gained by using the separate naming context to store DNS records. Select the `All domain controllers in the domain` option.

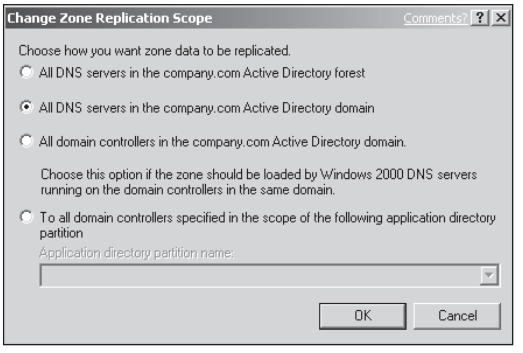

Figure 6.10 Change Zone Replication Scope window.

If you have a multidomain forest, use the `All DNS servers in the domain` option when you want to limit the scope of replication to a particular domain. This is typical for most domain-based zones.

All domain controllers in a forest need `SRV` and `CNAME` records from the zone representing the forest root domain. Under normal circumstances, DNS servers in the other domains would obtain these records recursively from the DNS servers in the root domain. You can speed this process up a little by setting the replication scope of the root domain to `All DNS servers in the forest`.

If this seems like too many records to replicate globally, you can create a new zone just for the resource records that require forest-wide scope. These records are stored in the forest root zone under `_msdcs`. For example, if the forest root domain were `Company.com`, you could create a new zone called `_msdcs.company.com`. The records would be extracted from the `company.com` zone and placed in this new zone. Set the replication scope for the `_msdcs.company.com` zone to `All DNS servers in the forest`.

Configuration Naming Context

The Configuration naming context holds information about parameters that control the services that support Active Directory. Every domain controller in a forest hosts a read/write copy of the Configuration naming context. It holds eight top-level containers. Here is a brief description of their purpose and content.

Display Specifiers

This container holds objects that alter the viewable attributes for other object classes. This is called *shadowing*. For example, the *User-Display* object shadows the User class. Display Specifiers provide *localization* and *context menu* functions.

Localization is the task of producing foreign language versions of an application. Rather than translate the contents of each attribute for each AD object into French, Italian, German, Spanish, Cyrillic, Kanji, Szechwan, Arabic, Korean, Hebrew, Thai, and so on, the system looks to see which country code was used during installation and filters the output through the appropriate Display Specifier.

Display Specifiers also define separate context menus, property pages, and icons based on whether or not the user accessing the object has administrator privileges. For example, when you right-click an object, the flyout menu that appears comes from a context menu associated with that object class. The Display Specifier filters the menu to display only those items you are permitted to perform.

Sorting Through Display Specifiers

When you view the contents of the DisplaySpecifiers container in Active Directory, you'll see a container with a number. This is the *code page* for the *National Language Group* in hex. The United States English code page is number 1033, which corresponds to 409 hex. The code pages for FIGS countries are French, 1036; Italian, 1040; German, 1031; and Spanish, 1034.

Extended Rights

Directory objects are also Windows security objects. This makes it possible to assign permissions to the object itself as well as any of the properties associated with the object. A User object can have many properties. Selecting precisely which properties to assign access rights to get a particular result can get tedious.

Extended Rights control access to objects by consolidating sets of property permissions into a single entity. For example, an extended right called Membership grants the ability to modify the membership of a single group, selected groups, every group in a container, or every group in a container and its subordinate containers.

Like the Display Specifiers mentioned previously, each Extended Rights object is associated with a structural object that it controls. For example, the *Personal-Information* and *Public-Information* objects are associated with both User and Contact classes.

There are over 50 Extended Rights objects covering a wide assortment of management operations, such as changing passwords, changing domain configurations, resetting user lockouts, and managing BackOffice services.

Lost and Found Config

This container holds objects that get orphaned during database replication. For instance, if a container is deleted during the same replication cycle that an object was created in the container, the object is sent to Lost and Found. Both the Domain and Configuration naming contexts have a Lost and Found container. The Schema naming context does not need one because Schema objects can never be deleted. See Chapter 7, "Managing Active Directory Replication," for more information.

Partitions

This container holds the cross-reference objects that list other domains in the forest. Domain controllers use the contents of this container to build referrals to other domains.

The Partitions container is extremely important for maintaining the integrity of a forest. It would be very bad to have objects representing invalid naming contexts in this container. For this reason, only one domain controller in a forest is permitted to update the contents of this container.

Physical Locations

This container holds Physical Location DN objects associated with Directory Enabled Networking (DEN). For example, a DEN-aware router can place a locator object in this container. Because DEN makes use of standard LDAP functionality, this is the only object class in Active Directory that uses the Location attribute.

The DEN initiative has developed a set of policies for controlling network parameters affecting Quality of Service (QoS), IP Security (IPSec), and other core networking functions. All leading routing and infrastructure vendors have pledged

support for DEN, and many have allied themselves with both Microsoft and Novell for the Directory part of DEN. Visit the web site of your favorite vendor to see its DEN-aware products and find out its plans for Active Directory integration.

Services
This container is exposed in the AD Sites and Services console by selecting VIEW | SHOW SERVICES option from the menu. Think of the contents of the Services container as a kind of enterprise-wide Registry. Distributed applications put objects into this container where they can be seen by other servers running the same application.

A disadvantage to this container is that it is replicated to every domain controller in the forest. You may have applications that only need their objects to be seen at selected domain controllers. For this reason, Microsoft included the ability to create a separate Application naming context that can be placed on individual domain controllers of your choice.

Sites
The Sites container is also exposed in the AD Sites and Services console. The objects in this container control Active Directory replication and other site-specific functions. Sites are used to control replication between domain controllers.

Well-Known Security Principals
The object-based security used by classic NT and Windows Server 2003 assigns a unique Security Identifier (SID) to every security principal. There is a set of well-known SIDs that represents special-purpose groups. This includes groups like Interactive, which designates users who are logged on at the console of a machine; Network, which designates users who have logged on to the domain; and Everyone, which designates every user. This container holds the names and SIDs of these groups.

Active Directory Trees and Forests

Recall that domains represent security boundaries for users as well as management boundaries for administrators. Users in one domain cannot access resources in another domain unless some provision is made to support a secure connection. If you have separate domains, you need a way to connect them into a single security structure. Classic NT uses Master domains and Resource domains for this purpose. Active Directory uses trees and forests.

Trees

Active Directory uses DNS domains to define its namespace. As we've seen, a standard LDAP hierarchy conforms to a contiguous namespace called a Directory Information Tree. An Active Directory namespace that follows a contiguous namespace is also called a *tree* (see Figure 6.11).

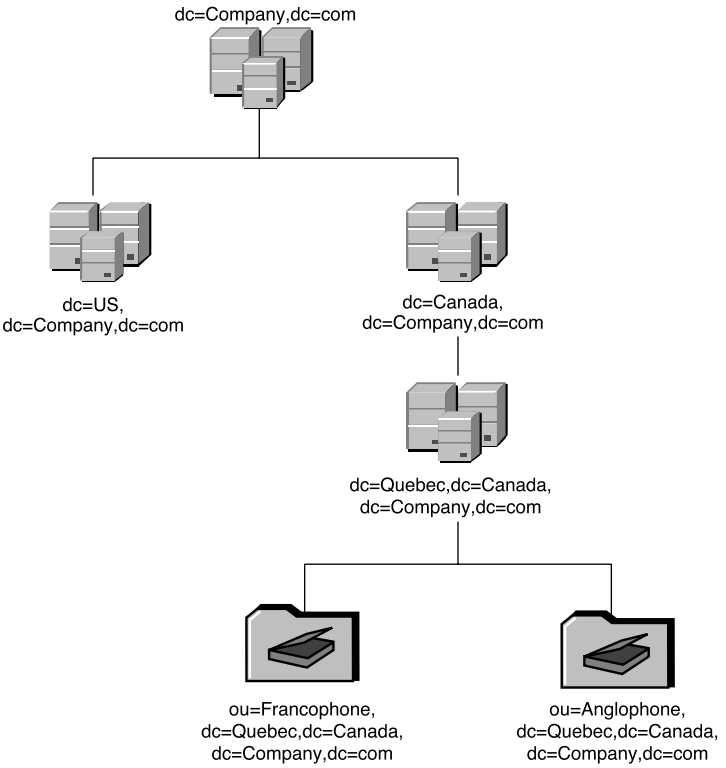

Figure 6.11 Active Directory tree.

Figure 6.11 shows the way an Active Directory tree coincides with a standard DNS namespace. In this diagram, a root domain called `Company.com` has two child domains, one for the `US` and one for `Canada`. The `Canada` domain has a child domain of its own for `Quebec`. The `Quebec` domain has Organizational Units (OUs) that divide objects depending on language. The `US` domain has OUs that divide objects depending on geography. Both represent acceptable uses of OU containers.

From an LDAP perspective, this tree structure looks pretty standard. If a client in the `Quebec` domain queries LDAP for information about a server in the `US` domain, the client will get a chain of referrals that walks the tree up to root and then down to a domain controller in the `US` domain.

Recall that each of these domains represents the contents of a naming context. The naming context for a domain is hosted on a domain controller in that domain. When a query walks the tree, it moves from one domain controller to another. If the domain controllers are in different locations in a WAN, the transaction may take a while to complete.

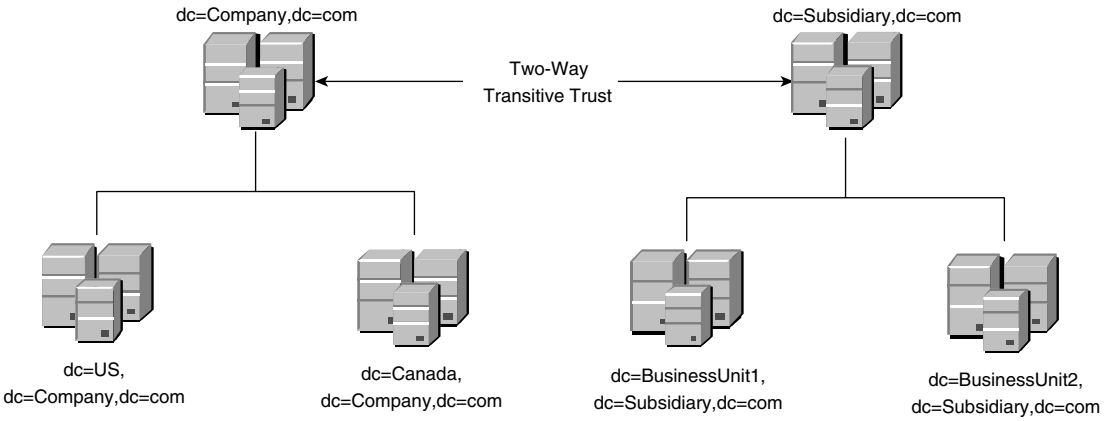

Figure 6.12 Active Directory forest.

Forests

Not every organization is fortunate enough to have a clean tree structure. Many companies have business units that are virtually autonomous fiefdoms with their own DNS root domains and independent administrative staffs and even separate lunchrooms. Many universities, too, have colleges with separate IT staffs and campuses that maintain their own infrastructures.

To accommodate these and other untree-like business structures, Microsoft tweaked the LDAP standard just a bit to develop a second structure called a *forest*. See Figure 6.12 for an example.

Domains in an Active Directory forest do not need to follow a contiguous namespace. A secure connection between the root domains forms a conduit that permits access by users in one domain to resources in the other domains.

Global Catalog

In a standard LDAP search involving multiple naming contexts hosted by multiple servers, the servers pass referrals to the client, and the clients walk the tree to get information from the various servers. This process of query and referral consumes time and bandwidth. And if one of those domain controllers is at the wrong end of a 56K line oversubscribed with users downloading MP3s, the search might take a while.

Active Directory speeds up searches and reduces WAN traffic by aggregating the contents of the various Domain naming contexts into a structure called a *Global Catalog*, or GC.

Global Catalog Structure

Because the GC contains a copy of every Domain naming context in a forest, it holds a copy of every object. In a big organization, this could make the database very large. It would not make sense to use separate domains to get separate naming contexts only to roll them up again into a GC that must be available at each location.

To reduce GC size and replication traffic, only a small number of commonly used attributes are stored in it. The list of attributes included in the GC is determined by the *Partial Attribute Set,* or PAS. The PAS contains only 200 or so out of the 1700 attributes in the Active Directory schema. Further, the partial naming contexts hosted by a GC server are read-only, so the GC server need only concern itself with replicating updates from a domain controller hosting a full copy of the naming context.

The Global Catalog is not a separate entity. A domain controller does not have a separate DIT file for a GC. Rather, the GC is really just a name for a domain controller function. The function is controlled by a flag in Active Directory. With the flag set to FALSE, a domain controller hosts only the standard three naming contexts—its own Domain NC, the Configuration NC, and the Schema NC. With the flag set to TRUE, the domain controller adds a partial replica for the other naming contexts in the forest. These naming contexts are stored in Ntds.dit right along with the three standard naming contexts.

Global Catalog Function

A Global Catalog server differentiates itself from a standard domain controller by listening for LDAP queries on a second port. The standard LDAP wire protocol uses TCP/UDP port 389. Global Catalog servers listen on this port but they also listen and respond on TCP/UDP port 3268. Here are the three possibilities for handling a search submitted to a GC on port 3268:

- If the GC server receives a search request involving an attribute or attributes in the Partial Attribute Set (PAS), it responds to the request with a dataset containing the requested objects and attributes.
- If the GC server receives a search request involving an attribute or attributes that are not in the PAS but the objects are in its own domain, it responds to the request with a dataset containing the requested objects and attributes. It obtains this information from the full copy of its Domain naming context.
- If the GC server receives a search request involving an attribute or attributes that are not in the PAS and for objects in another domain, it responds to the request with a referral to the other domain. The LDAP client follows up on the referral and completes the search by walking the tree.

Global Catalog servers play a crucial role in the operation of Active Directory in an enterprise. If you have a multidomain forest, it is very important that all users be able to reach a GC server. In the next few chapters, we'll come back to the operation of the GC and the role it plays in authentication and access control.

Global Catalogs and Naming Context Locations

Just as a recap, take a look at the forest in Figure 6.12. The forest contains six different domains. Let's say that a domain controller in the `Canada` domain is configured to be both a Global Catalog server and a DNS server for an Active Directory Integrated DNS zone. This server would host the following naming contexts:

- A full, read/write copy of `dc=Canada,dc=Company,dc=com`
- A full, read/write copy of `cn=Configuration,dc=Company,dc=com`
- A full, read-only copy of `cn=Schema,cn=Configuration,dc=Company,dc=com`
- A full, read/write copy of `cn=ForestDNSZones,dc=Company,dc=com`
- A full, read/write copy of `cn=DomainDNSZones,dc=Canada,dc=Company,dc=com`
- A partial, read-only copy of the remaining domain naming contexts

Active Directory Trust Relationships

Creating trees and forests requires a way to pipe secure transactions between the various domains. Like classic NT, this pipe is called a trust relationship.

Classic NT trusts have always reminded me of the Dr. Seuss story *The Zax*. In this story, a North-going Zax meets a South-going Zax in the prairies of Prax. They stand nose to nose, unwilling to move out of each other's way. One Zax says, "I'm a North-Going Zax and I always go north. Get out of my way, now, and let me go forth!" To which the other Zax replies, "You're in MY way! And I ask you to move, and let me go south in my south-going groove."

A classic one-way NT trust behaves exactly the same way. The rights implicit in the trust only flow in one direction and cannot flow from one domain to another. A big NT system based on interlocking trusts between many different resource and master domains can begin to look a little like a Dr. Seuss drawing, as well.

Transitive Kerberos Trusts

The picture gets a little neater with Active Directory. When two Active Directory domains trust each other, users and groups and computers from one domain can seamlessly access resources in the other domain. The trust flows both ways.

In addition, Active Directory trusts are transitive, meaning that they flow from one domain to another if domains are chained together. For example, if five Active Directory domains trust each other, users from one domain can access resources in any of the other four domains, assuming that they have been granted access permissions.

The magic that makes this work comes from the Kerberos authentication mechanism that underlies the trust relationships. See Chapter 12, "Managing Group Policies," for more information about how Kerberos works and how it supports transitive trusts.

Trust Types

Active Directory domains have several ways they can trust each other, or trust down-level NT domains, depending on the structure you want to build. There are six types of trust relationships, illustrated in Figure 6.13:

- **Parent/Child trusts.** This style of trust exists between two Active Directory domains that share a contiguous DNS namespace and belong to the same forest.
- **Tree Root trusts**. This style of trust exists between root domains in the same forest that do not share a common DNS namespace.
- **Shortcut trusts**. This style of trust exists between two domains in different trees within the same forest. It is used to expedite Kerberos transactions between the domains. With a shortcut trust in place, a client can obtain a Kerberos ticket directly from the trusted domain without walking the tree.

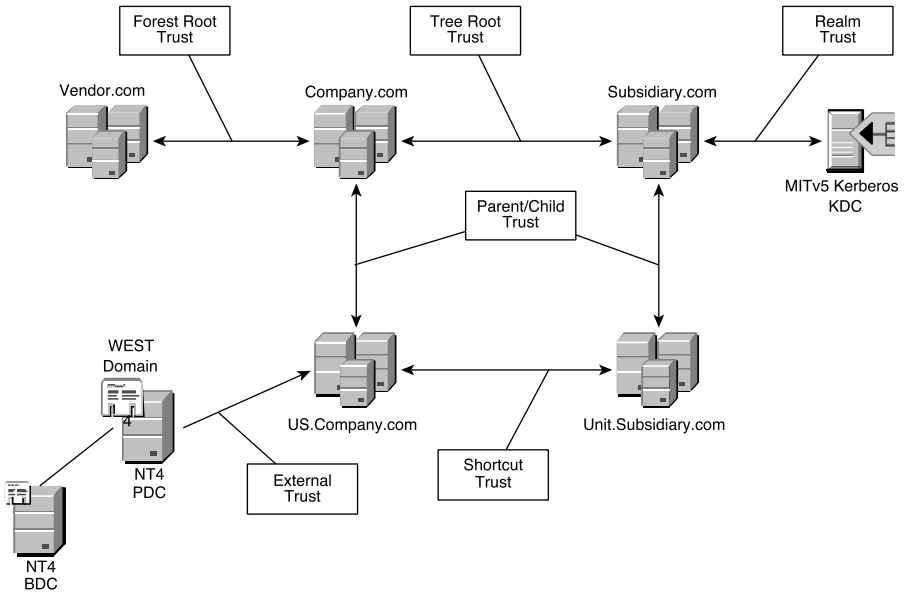

Figure 6.13 Diagram of various Active Directory trust options in Windows Server 2003.

- **External trusts.** This style of trust exists between an Active Directory domain and a downlevel NT4 domain. You can also create an external trust to a Samba domain. An External trust resembles a classic NT trust. It is one-way and non-transitive, meaning it cannot link an entire forest to a downlevel domain. LDAP searches and Kerberos authentications do not cross the trust boundary.
- **Kerberos realm trusts.** This style of trust exists between an Active Directory domain and an MIT v5 Kerberos realm. (MIT stands for Massachusetts Institute of Technology, where Kerberos originated.) The trust can be made transitive and two-way.
- **Forest trusts**. This style of trust exists between two Active Directory forests. It can be made transitive and two-way. The forests do not share a common Schema or Configuration naming context. This trust type forms a *federation* of forests. It is used to join two organizations that have existing Active Directory deployments and do not want to migrate accounts into a single forest. This trust type is a new feature in Windows Server 2003 and requires all domains in the forests and the forests themselves to be running at full Windows Server 2003 functionality (no legacy domain controllers.) You must be a member of the Incoming Forest Trust Builders group to create a Forest trust.

The new Forest Trust type should not be considered a panacea for organizational restructurings. Because the two forests do not share a common Configuration or Schema naming context, they cannot share applications that require a common configuration. A principal example is Exchange 2000, which places critical information into the Configuration naming context. A federation of forests cannot be placed into a single Exchange organization, so users cannot see a common Global Address List (GAL) or use common distribution lists.

Establishing Parent/Child and Tree Root Trusts

Parent/Child trusts and Tree Root trusts can only be formed when a domain is created. There are no tools for consolidating domains (grafting) or prying them apart (pruning) after the forest is in place. Every domain controller in a forest hosts an identical copy of the Schema naming context. There are no tools to coordinate and consolidate two sets of schemas.

I'm going to repeat this point in a different way because it's important to remember. New domains can only be added to a forest when the first domain controller is promoted in that domain. After you create a forest, the constituent domains cannot be removed without demoting every domain controller in the domain, essentially losing all Active Directory information for that domain.

External and Realm trusts do not rely on the Schema or Configuration naming contexts, so they can be created and broken while leaving the end-point domains intact. If you break the trust, users in the trusted domain lose access to resources in the other domains. If you make the trust again, the users regain access.

Establishing Forest Trusts

Forest trusts are a bit more complex to configure than the standard Windows 2000 trust types. Although the trust itself is two-way and transitive, you can select the domains in each forest that participate in the trust. For example, consider the diagram in Figure 6.14. This diagram shows two forests in a federation connected by a Forest trust. Each forest has domains connected by Parent/Child trusts.

In a fully transitive configuration, users in the `US.Company.com` and `Canada.Company.com` domains would be able to access resources in the `PacRim.Subsidiary.com` and `Europe.Subsidiary.com` domains and vice-versa. However, you may not want to enable fully transitive resource access. Using the Properties window for the trust, you can select which domains will participate in the Forest trust and in which direction the trust will be effective.

Using this feature, you can target trusts in the federation. For example, you can configure the Forest trust so that users in `PacRim.Subsidiary.com` can access resources in `US.Company.com` but not in `Canada.Company.com`.

You must be a member of the Incoming Forest Trust Builders group to create a Forest trust to another domain. This new Builtin group permits a root domain administrator to grant permissions for an administrator in another root domain to create a trust without giving that administrator full domain administrative privileges.

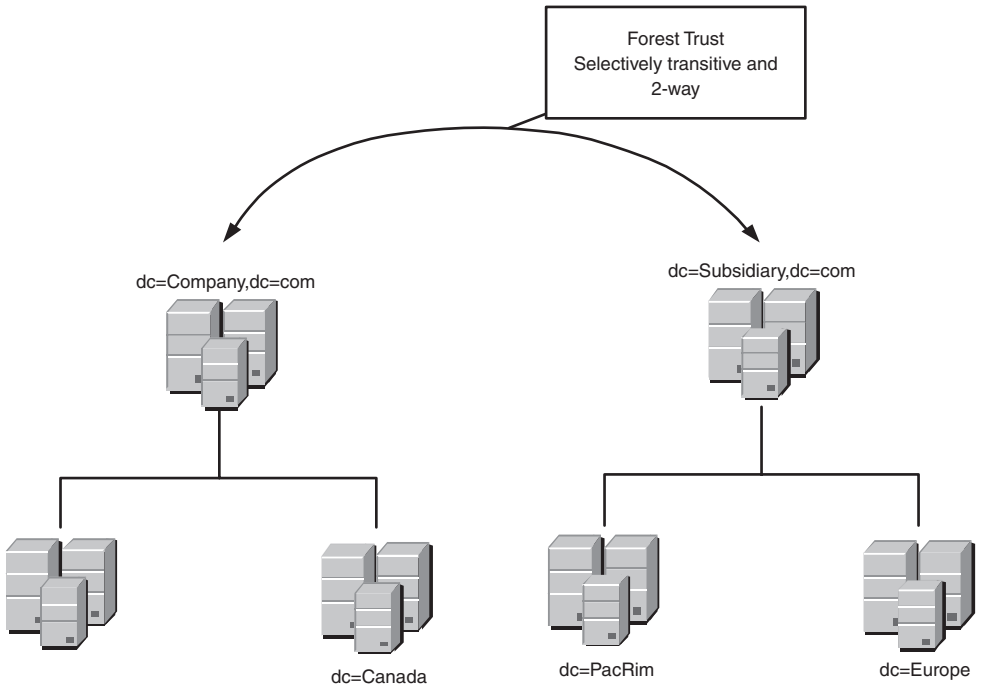

Figure 6.14 Federation of two Active Directory forests.

Object Migration Between Domains and Forests

You cannot build or break Parent/Child and Tree Root trusts after they are formed, so the only way to change your forest structure is to migrate objects between domains. Microsoft provides a utility for performing these object migrations called the *Active Directory Migration Tool*, or ADMT.

Moving user, computer, and group accounts between domains involves issues of security and accessibility. Both classic NT and modern Windows servers use Security IDs (SIDs) to identify users. These SIDs are placed on access control lists (ACLs) to control access to resources. ADMT performs a complex set of functions designed to preserve the original SIDs so that users retain access to their resources. The new version in Windows Server 2003 also preserves passwords and the original user profiles. There are a variety of third-party tools that can help with object migration between domains and forests. See Chapter 9, "Deploying Windows Server 2003 Domains," for details.

Sysvol

There's more to being a domain controller than simply hosting the Active Directory database. The domain controller is also responsible for distributing the files associated with group policies. Group policies are used to centrally manage member servers, desktops, and users. They are covered in detail in Chapter 12, "Managing Group Policies."

Active Directory domain controllers must also support downlevel clients by providing a place to obtain classic scripts and system policies contained in Config.pol or Ntconfig.pol files. In an NT domain, these files are stored in the Netlogon share, physically located at `C:\Winnt\System32\Repl\Import\Scripts`.

Sysvol Files

To meet its dual responsibilities of supporting modern group policies and classic system policies and scripts, Active Directory domain controllers host a special folder called Sysvol. The location of the folder is determined during Dcpromo. Sysvol must be on an NTFS volume because folders within Sysvol use reparse points, which are only supported by NTFS.

Sysvol contains a folder with the name `Domain` that holds the group policy files in a folder called `Policies` and classic scripts in a folder called `Scripts`. The `Scripts` folder is shared as Netlogon to support downlevel clients. Modern scripts that are distributed as part of group policies are stored as part of a particular group policy under the `Policies` folder.

Clients access Sysvol via a special *fault tolerant share* with the Universal Naming Convention (UNC) path of `\\<domain_name>\Sysvol`. For example, you can do a directory of `\\company.com\Sysvol` from any client in the `Company.com` domain. Accessing fault tolerant shares requires that the Dfsclient service be running on the client.

File Replication and Sysvol

The contents of Sysvol are replicated to every domain controller in a domain. It is important that the contents stay in sync. Otherwise, users will get different group policies, system policies, and classic scripts when they log on to different domain controllers.

A service called the *File Replication Service*, or FRS, is responsible for synchronizing the contents of Sysvol between domain controllers. (The actual service name is Ntfrs, which you may see in Event log entries.) FRS replicates an entire file when any changes are made to the file. To prevent race conditions that could occur if the file were locked, the file is first copied to a `Staging` folder then replicated to the other domain controllers.

Locating Active Directory Services

Active Directory clients use DNS to locate domain controllers. They do this by querying for Service Locator (SRV) records that point at LDAP, Kerberos, and Global Catalog ports on the servers. Refer to RFC 2052, "A DNS RR for Specifying the Location of Services." (RR stands for Resource Record.)

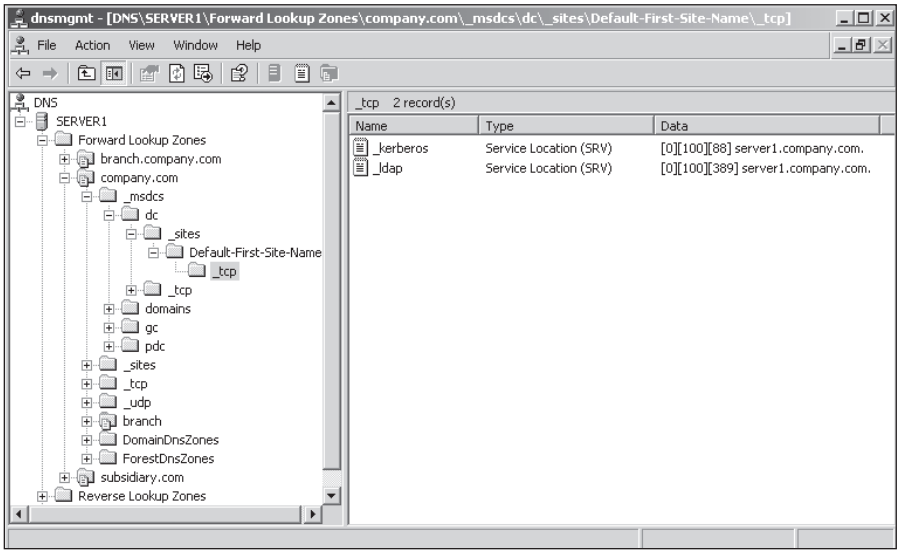

Figure 6.15 DNS console showing SRV records for the `Company.com` domain.

SRV Records for Active Directory

Figure 6.15 shows a DNS zone table for the `Company.com` domain. The zone table contains the SRV records registered by the first domain controller in a Windows Server 2003 domain.

Here are the SRV records in a standard zone table format:

```
_kerberos._tcp.phoenix._sites.dc._msdcs    600     SRV     0 100 88
➥dc-01.company.com._
 kerberos._tcp.phoenix._sites              600     SRV     0 100 88
➥dc-01.company.com.
_kerberos._tcp.dc._msdcs                   600     SRV     0 100 88
➥dc-01.company.com.
_kerberos._tcp                             600     SRV     0 100 88
➥dc-01.company.com.
_kerberos._udp                             600     SRV     0 100 88
➥dc-01.company.com.
_kpasswd._tcp                              600     SRV     0 100 464
➥dc-01.company.com.
_kpasswd._udp                              600     SRV     0 100 464
➥dc-01.company.com.
_ldap._tcp.phoenix._sites.gc._msdcs        600     SRV     0 100 3268
➥dc-01.company.com.
_gc._tcp.phoenix._sites                    600     SRV     0 100 3268
➥dc-01.company.com.
_ldap._tcp.gc._msdcs                       600     SRV     0 100 3268
➥dc-01.company.com.
_gc._tcp                                   600     SRV     0 100 3268
➥dc-01.company.com.
_ldap._tcp.phoenix._sites.dc._msdcs        600     SRV     0 100 389
➥dc-01.company.com.
_ldap._tcp.phoenix._sites                  600     SRV     0 100 389
➥dc-01.company.com._
 ldap._tcp.dc._msdcs                       600     SRV     0 100 389
➥dc-01.company.com.
ldap._tcp.{guid of domain}.domains._msdcs  600     SRV     0 100 389
➥dc-01.company.com.
_ldap._tcp                                 600     SRV     0 100 389
➥dc-01.company.com.
_ldap._tcp.pdc._msdcs                      600     SRV     0 100 389
➥dc-01.company.com.
 dc-01                                     1200    A     10.1.1.1
 gc._msdcs                                 600     A     10.1.1.1
 {GUID of DC invocation}._msdcs              600     CNAME
➥dc-01.company.com.
```

Format of SRV Record Names

The leading underscores in SRV record names are there to avoid collision with other records by the same name. The naming format is specified in RFC 2052, "SRV Record Format and Use."

Windows DNS reverses the SRV record names to display them as a hierarchy of folders. Here are the functions of the SRV records based on their groupings in the DNS console:

- **_MSDCS.** This heading collects SRV records based on their status as domain controllers, domain invocations, global catalog servers, and primary domain controllers. Domain controllers and global catalog servers are broken down by site. This tells Active Directory clients very quickly where to find local services. Domain invocations support replication. Each domain controller gets a GUID that it uses when invoking replication. The PDC entry contains the SRV record for the domain controller assigned to be the PDC Emulator, a domain controller that acts as the PDC to downlevel NT BDCs.
- **_SITES.** A site represents an area of high-speed connectivity associated with one or more distinct IP subnets. By indexing domain controllers based on their site affiliation, clients can look in _SITES to find local services rather than sending their LDAP lookups across the WAN. Standard LDAP queries use port 389. Global Catalog queries use port 3268.
- **_TCP.** This heading collects all domain controllers in the DNS zone. The _TCP grouping acts as a catchall for clients that cannot find their specific site or that need to find a domain controller elsewhere in the network if none of those with local SRV records respond.
- **_UDP.** Kerberos v5 permits clients to use connectionless services to get tickets and change passwords. This is done via UDP ports that correspond to the TCP ports for the same services, UDP port 88 for ticketing and UDP 464 for password changes.

Operational Description of SRV Record Queries

When a user initiates a process that requires an Active Directory lookup, the AD client process sends a query to DNS for SRV records corresponding to server advertising LDAP ports. The first query is for SRV records in the client's local site. This ensures that LDAP searches do not go to domain controllers elsewhere in the WAN. If there are no domain controllers in the client's site, it asks for all SRV records regardless of site.

Registry Tip: Site Name Cache

Clients cache their site information in the following Registry location:

```
Key:     HKLM | System | CurrentControlSet | Services | Netlogon | Parameters
Value:   DynamicSiteName
Data:    Flat name of the last domain controller authenticating the client—
for example, dc-01
```

DNS returns all SRV records that meet the query conditions. If there are five domain controllers in a site, DNS returns five SRV records accompanied by the Host record containing the IP address of the server in each SRV record. This is different than standard DNS operation, where it would normally return a single record in a round-robin selection process.

When the client receives the SRV records, it performs a quick LDAP ping to all of them by sending out a bind query to UDP port 389. The first domain controller to respond is selected as the primary LDAP server by the client. Here are details of the transaction:

1. When the operating system loads, the network client locates a domain controller by querying DNS for SRV records. The client in the diagram sends a query for `_kerberos._tcp.Phoenix._sites.dc._msdcs.Company.com`. Notice that the scope of this query is limited to domain controllers from the same site and domain. The client stores the site name in the Registry under `HKLM | System | CurrentControlSet | Services | Netlogon | Parameters | DynamicSiteName`.
2. When the DNS server receives this query, it returns all SRV records that meet the query criteria, sorting them by priority and weight.
3. When the network client receives the SRV records, it fires off an LDAP ping (a single UDP packet) over port 389 to every domain controller on the list. It sends these pings in rapid succession, every one-tenth of a second.
4. When a domain controller gets the LDAP ping, it returns an LDAP response. The client designates the first domain controller to respond as the logon server and proceeds to authenticate via Kerberos.

At this point, the client behaves like a lonely kid who has finally found a friend. It hounds the domain controller with all subsequent LDAP requests, Kerberos authentication requests, and group policy downloads.

You can determine the identity of the domain controller that authenticated a member Windows Server 2003 using the SYSTEMINFO utility. Here is a partial listing showing the logon server information:

```
Virtual Memory: Max Size:  1,733 MB
Virtual Memory: Available: 1,344 MB
Virtual Memory: In Use:    389 MB
Page File Location(s):     C:\pagefile.sys
Domain:                    company.com
Logon Server:              \\DC01.company.com
Hotfix(s):                  0 Hotfix(s) Installed
```

If the client is in a forest, the domain controllers generate referrals to other domains. Clients use SRV records for those domains to locate domain controllers that host copies of the target Domain naming contexts.

Site Coverage

You cannot configure a preferred domain controller for a client. If you have a large LAN and you want to compartmentalize your clients based on their area of a campus LAN or MAN (metropolitan area network), you must structure your replication topology around multiple sites. This is true even if your WAN interties meet the requirements for a high-speed connection that would not normally require separate sites.

Domain controllers automatically register their SRV records using their site name. They also return referrals to clients to ensure that clients use a local domain controller for authentication and LDAP queries. This "localization" feature is possible because each site is associated with one or more IP networks connected by Site Links. A domain controller can read the IP address of a client and determine the site it should designate when making DNS requests for SRV records.

Here's how this works. Let's say that the client is a laptop. The user shuts down the laptop, flies to Houston, and connects to the network again:

1. The client gets a local address from Dynamic Host Configuration Protocol (DHCP). It remembers that it is in the Phoenix site and queries DNS for domain controllers in that site.
2. DNS returns the requested SRV records and the client sends LDAP pings to the domain controllers in Phoenix.
3. A domain controller in Phoenix examines the client's IP address and sees that the client is in the Houston site. It knows this by comparing the IP address to the IP Subnet objects in Active Directory.
4. The domain controller responds with a referral telling the client to query DNS for the Houston site.
5. The client responds by repeating the DNS query for SRV records from the Houston site. In this way, the client automatically adjusts to changes in location.

Clients cache their site information in the following Registry location:

```
Key:   HKLM | System | CurrentControlSet | Services | Netlogon | Parameters
Value: DynamicSiteName
Data:  Flat name of domain controller authenticating the client—for example,
dc1
```

This site localization feature herds clients toward local domain controllers for authentication and LDAP queries. If you have a location that does not have a domain controller, you should still create a site for the location. This populates DNS with SRV records for the next site upstream so that clients authenticate at the closest domain controllers.

Compatibility Settings

For backward compatibility, certain Active Directory features are disabled while domain controllers running something other than Windows Server 2003 are in operation.

A Windows Server 2003 domain faces two compatibility challenges (at least with other Windows servers):

- Operation with downlevel NT domain controllers
- Operation with Windows 2000 domain controllers

Each of these challenges requires a different compatibility setting.

Operation with Downlevel NT Domain Controllers

Active Directory domain controllers can coexist with NT4 Domain Controllers in the same domain. This is called *Windows 2000 Mixed*. In Mixed, a Windows Server 2003 domain controller designated as the PDC Emulator uses classic LMRepl (LanMan Replication) to deliver selected Active Directory updates to downlevel BDCs.

In Mixed, certain advanced features in Active Directory are disabled because they are incompatible with classic NT4. Here is a list:

- **Universal groups.** This group type can have members from any domain in a forest and can be placed on access control lists anywhere in a forest.
- **Global group nesting.** In Native, Global groups from different domains can be nested together and nested into Universal groups.
- **Local access to Domain Local groups.** In Native, Domain Local groups from Active Directory can be placed on access control lists on member servers and desktops.
- **Downlevel clients can participate in transitive authentication.** After a domain is running in Native, the domain controllers can proxy NTLM authentication requests from downlevel clients to give them access to domains that they would not be able to access in a standard NT master/resource domain structure.

After you have upgraded or decommissioned all NT4 BDCs, you can get these advanced features by shifting the domain to *Windows 2000 Native*. This stops replication from the PDC Emulator to any remaining NT4 BDCs. After a domain has been shifted to Windows 2000 Native, it cannot be shifted back to Windows 2000 Mixed.

Functional Levels

Several new Windows Server 2003 features are incompatible with Windows 2000. Here is a quick list:

- The calculations for determining replication topology between sites have been streamlined. This corrects a problem where large organizations with hundreds of sites might experience replication failure because the topology calculations could not be completed in the time allotted to them.
- Group members are now replicated as discrete entities instead of replicating the entire group membership list as a single unit. This corrects a problem where membership changes made to the same group on different domain controllers in the same replication interval would overwrite each other.
- A new trust type has been added to simplify transitive trust relationships between domains that are not in the same forest.
- Support has been added for the inetOrgPerson object class, which is used on other commercial LDAP directory services to represent users. inetOrgPerson objects can be given a SID and used as security principals for logon and put on access control lists.
- Domain controllers can be renamed in a Windows Server 2003 domain. Domains themselves can be renamed in a Windows Server 2003 forest. This permits restructuring a forest by changing parent/child relationships between domains.
- Schema objects can be declared defunct so that the parameters can be reused in another Schema object. A Schema object cannot be deleted nor can the Common Name (CN) be changed.
- Changes made to elements of the Global Catalog, such as adding an attribute to the GC or taking one away, do not now require a full rebuild and replication of the GC.

As long as Windows Server 2003 domain controllers coexist with Windows 2000 Domain Controllers, these features are disabled. When all Windows 2000 Domain Controllers have been upgraded to Windows Server 2003 or demoted to standard servers, the domains and the forest can be shifted to full Windows Server 2003 functionality. This is a one-time operation and cannot be reversed. See Chapter 9, "Deploying Windows Server 2003 Domains," for the prerequisites and steps to change functional levels.

Client Compatibility

Windows Server 2003 Active Directory domains are compatible with any and all Windows clients as well as the Microsoft DOS client and the most current versions of Samba.

The opposite is also true. Windows Server 2003 and XP can operate in any Windows domain environment: classic workgroups, classic NT, Windows 2000 Active Directory, and of course, Windows Server 2003 Active Directory. (The sole exception is XP Home Edition, which cannot join a domain of any form.)

One subtle problem that arose in Windows 2000 was fixed in Windows Server 2003 and in Windows 2000 SP2. When Kerberos-based Windows clients operate in downlevel domains, they happily use NTLM Challenge-Response for their authentication. This means they can log on to classic backup domain controllers (BDCs) and participate in pass-through authentication.

Piling On

When the domain is upgraded to Active Directory, however, a Kerberos-based client changes a flag in its security database to disable NTLM Challenge-Response and use only Kerberos. This means that if you have deployed a few thousand Windows 2000 or XP desktops in your NT domain, as soon as you upgrade the PDC, all those desktops will scurry to that one machine to authenticate. Microsoft calls this behavior "piling on."

In addition, after a client has authenticated with an Active Directory domain controller, it behaves like a teenager who has finally gotten up the gumption and money to move out of the house. It sets a flag in its local security database and thereafter will only authenticate with Active Directory domain controllers. If only classic BDCs are available, the client logs users on using cached credentials rather than deign to use a classic BDC. This can cause operational difficulties if you have large numbers of desktops and member servers that have already been upgraded to Windows 2000 or XP or Windows Server 2003 when you do the upgrade of the PDC. If the clients are in Guam and your PDC is in Galveston, the morning logons in Guam are going to be exceedingly slow.

To avoid this problem, Windows Server 2003 includes a feature that keeps an Active Directory domain controller pretending that it is still a downlevel domain controller to its clients. After you have installed enough Windows Server 2003 Domain Controllers to handle the logon requests, you can pull up the curtain and turn on the footlights and let the clients switch to Kerberos authentication.

The feature consists of a Registry entry that makes a newly promoted Windows Server 2003 domain controller pretend to be classic NT4 domain controller. Here is the entry:

```
Key:    HKLM | System | CurrentControlSet | Services | Netlogon | Parameters
Value:  NT4Emulator
Data:    1 (REG_DWORD)
```

It is important that you put this entry in place on all NT domain controllers *before* you upgrade them. The domain controller will still register its SRV records, but when the modern Windows clients go to authenticate, the domain controller will only respond with an NTLM authentication sequence.

Special NT4 Emulator Considerations

During the time that you have the `NT4Emulator` switch in place, your XP and Windows 2000 desktops will continue to use NTLMv2 authentication rather than Kerberos. This imposes the following limitations:

- Clients do not download or implement group policies.
- You cannot use Active Directory management tools such as AD Users and Computers or AD Sites and Services from the client because it has not authenticated using Kerberos and therefore cannot gain LDAP access to Active Directory.
- You cannot promote a member server to a domain controller because it cannot make LDAP connection to an existing domain controller.
- If the `NT4Emulator` switch is set on domain controllers in the root domain of the forest, you cannot create a new domain in the forest because the new domain controller cannot make LDAP connection to an existing domain controller in the root domain.

You can avoid these limitations on a case-by-case basis by permitting the client to ignore the `NT4Emulator` behavior of a domain controller and to log on using Kerberos. Do this by putting an entry into the Registry at the client:

```
Key:    HKLM | System | CurrentControlSet | Services | Netlogon | Parameters
Value:  NeutralizeNT4Emulator
Data:   1 (REG_DWORD)
```

After putting this entry in place, log off and back on again. The desktop finds the Windows Server 2003 domain controller and uses Kerberos to authenticate. You can verify that this occurs using the Kerbtray utility from the Resource Kit.

When you have sufficient Windows Server 2003 domain controllers deployed to handle the expected volume of Kerberos authentications and group policy deliveries, flip the `NT4Emulator` switch to 0 in the Registry of each domain controller and restart it. This enables the domain controller to authenticate using Kerberos as well as NTLMv2. Be sure you flip the switch on all domain controllers to avoid confusion.

Active Directory Namespace Highlights

Here is a summary of the key points to remember about how the Active Directory namespace is structured. These points become critical design elements when the time comes to deploy Active Directory in your organization:

- The Active Directory database is divided into separate replication units called naming contexts. There are four types of naming contexts: Domains, Configuration, Schema, and Application.
- Active Directory domains form separate security and management units as well as separate naming contexts.

- Every domain controller in a forest has a replica of the Configuration and Schema naming context. This ensures that the domain controllers share the same knowledge about Active Directory topology, operation, and object management.
- Separate Active Directory domains can be connected together into a common security structure. If the domains share a contiguous DNS namespace, they form a tree. If they do not share a contiguous namespace, they form a forest.
- Active Directory uses trust relationships between domains to form trees, forests, and secure connections to external domains, forests, and MIT Kerberos realms. A trust can also be used to create a shortcut between domains in the same forest.
- Trust relationships between Kerberos-based Windows domains can be made transitive and two-way. Trusts to downlevel domains are one-way and non-transitive.
- Active Directory improves the performance of deep LDAP searches (searches that include multiple domains) by aggregating a partial replica of all Domain naming contexts into a Global Catalog. Any domain controller can host a copy of the GC.
- Active Directory clients use SRV records in DNS to locate Active Directory services on domain controllers. Clients preferentially use domain controllers from their local network to reduce WAN traffic and improve performance.
- Windows Server 2003 maintains backward compatibility to both classic NT4 domains and Windows 2000 domains.
- All domains in a forest and the forest itself must be shifted to Windows Server 2003 Functional Level to get access to all new Active Directory features.

Active Directory Schema

When discussing directory service structure and operation up to this point, I've used general terms that are applicable to just about any LDAP implementation. It's now time to spend a while looking at specific features in Active Directory. You may find this information to be a little too much detail for helping you manage day-to-day operations in Windows Server 2003. However, it's good to know some of the important functional and operational details of the directory service to help you create reliable domain designs and to troubleshoot problems that arise.

As a quick review, the object-oriented LDAP database that comprises Active Directory is structured around a set of object classes and their associated attributes. Individual objects are instances of specific object classes. The schema defines the available object classes, their associated attributes, the data types and permitted ranges for those attributes, and the rules for arranging and managing objects within the Active Directory database.

Schema Functional Operation

To visualize how the schema works, consider a simple, paper-based directory. Every month or so I get a catalog from Land's End, the clothing retailer. This catalog is a database of sorts, similar to a directory service except that it guides the user to a garment instead of a network entity. Consider this:

- The schema for this directory defines a set of object classes with the scope "Garments Sold by Land's End." These classes represent objects of interest to garment purchasers, such as Sweaters, Suits, Blazers, Accessories, and so forth.
- The schema also defines the available attributes that can be associated with the object classes, such as Size, Color, Inseam-Length, and Price, along with more subtle attributes specific to the directory itself, such as Picture-Of-Garment.
- The schema has *content rules* that define what attributes can be associated with a class. Some attributes, like Size and Color, might be associated with nearly every class. An attribute like Inseam-Length, however, might only be associated with classes like Slacks and Jeans, not Sport-Coats or Shoes.
- Some garment classes have attributes that are nearly identical. For example, the attributes that define the Polo-Shirts class differ only slightly from the attributes that define the Sport-Shirts class. The Polo-Shirts class derives from the Sport-Shirts class and inherits the attributes associated with its parent. The new attributes are then just tacked on to the new class.
- Class inheritance makes it important to have *structure rules* that keep the directory aligned with the real world. For example, a structure rule prevents placing an object from the Bathrobe class under a container from the Shoe class.
- A particular garment is an *instance* of its garment class. For example, an instance of the Blazer class would be the solid red blazer with green plaid lining that I gave my brother for Christmas last year. (The snide thank you note I received in return came from the Hallmark directory service as an instance of the Ungrateful-Sibling class.)
- The Land's End schema has *syntax rules* that define the values that can be associated with an attribute. For example, the Size attribute must have whole integer values while the Shoe-Size attribute can have real number (fractional) values.
- Because the garment classes and their attributes can change, the Land's End directory is *extensible*. For example, a new attribute called Number-Of-Sleeve-Buttons can be added and the Blazers class modified to include that attribute.
- For flexibility, certain special object classes can be *dynamically assigned* to a specific object. This makes it possible to create special bundles of attributes for a certain object like a Rad-Phat T-shirt object without altering all other instances of the T-shirt class.

I know this was a long example, so here are the key terms and concepts:

- **Object Classes.** Define the objects that can appear in Active Directory and their associated attributes.
- **Class Derivations.** Define a method for building new object classes out of existing object classes.
- **Object Attributes.** Define the available attributes. This includes extended attributes that govern actions that can be taken on object classes.
- **Structure Rules.** Determine possible tree arrangements.
- **Syntax Rules.** Determine the type of value an attribute is capable of storing.
- **Content Rules.** Determine the attributes that can be associated with a given class.
- **Extensible schema.** Additions can be made to the list of available classes and attributes.
- **Dynamic class assignments.** Certain classes can be dynamically assigned to a specific object rather than an entire class of objects.

Object Classes and Class Derivations

An object class is nothing more than a bundle of attributes with a name. The User class, for example, has certain attributes that, taken together, make it distinct from the Organizational-Unit class or the Server class. The X.500/9594 standard as modified by RFC 2256, "A Summary of the X.500(96) User Schema for use with LDAPv3," defines 21 classes and 55 attributes in a standard LDAP directory schema.

The Active Directory schema extends this list quite a bit, out to nearly 200 classes and just under 1700 attributes. If you want a complete list, check out the Windows Server 2003 Platform SDK or look at the MSDN web site, `msdn.microsoft.com`.

Standard LDAP Classes and Attributes in Active Directory

The Active Directory schema includes all RFC 2256 classes, except for Alias and Strong-Authentication-User, and all attributes, except for Aliased-Object-Name. The exclusion of Alias was deliberate. Aliases are a notorious source of performance difficulties and integrity problems in directory services. In addition, most of the object classes that would normally be given aliases are required to have unique names in Active Directory. This includes Users, Computers, and Groups.

Windows .NET includes the inetOrgPerson class as defined in RFC 2798, "Definition of the inetOrgPerson Object Class." This makes Active Directory more compatible with Netscape Directory Services and Novell Directory Services, both of which derive their User class from inetOrgPerson.

Schema Rules

It's not enough to define schema components in terms of objects, actions, and relationships. Laws and customs are also necessary to avoid anarchy. These take the form of *schema rules*. Directory service designers build certain rules into the schema that determine how classes and attributes are used, what kind of values they can have, and what relationship they have to each other. These rules fall into three categories:

- Structure Rules
- Content Rules
- Syntax Rules

Structure Rules

Frank Lloyd Wright established the design paradigm for twentieth century architecture by declaring that form should always follow function. He was a building architect rather than directory services architect, of course, but Active Directory is as much of a monument to form and function as a prairie house, and it is the *structure rules* that accomplish this.

There is really only one structure rule in Active Directory: Each object class has only certain classes that can be directly above it, called *Possible Superiors*. This structure rule is very important because classes inherit attributes from their parents. Structure rules prevent putting a User class object under a totally unrelated container class, like IPSEC-Base or NTDS Settings.

Content Rules

Every object class has certain attributes with values that cannot be left blank when an object is instantiated. These are called *must-contain attributes*. For example, every instance of the User class must have a value for the Common-Name attribute. Other attributes are optional and are designated *may-contain attributes*.

An important design principle of Active Directory is that only attributes with values are stored in the database. This greatly reduces the size and complexity of the database. Because attributes can be added after an object is created and then later removed if they are set to null, the database engine must constantly pack and repack the data. This is done by a garbage collection service that runs every 12 hours.

Syntax Rules

Attributes store data. Data must have a data type to define the storage requirements. Real numbers have a different form from integers, which are different from long integers, which are different from character strings.

An attribute can have only one data type. It cannot hold a string when associated with one object class and an integer when associated with another. The syntax rules in the schema define the permissible values types and ranges for the attributes.

Schema Definition Objects

Individual objects are always instances of an object class. Achieving this design principle involves using a template that defines the attributes, schema rules, and class hierarchy for the objects within an object class. The same applies for attributes, which require a template to define the syntax rules. This suite of templates makes up the schema definitions for a directory service information store.

Some directory services put the schema definitions into a separate file that is loaded at boot time or whenever the schema requires changing. In contrast, the Active Directory schema is self-referential. That is to say, all class definitions, attribute definitions, and schema rules are part of the schema itself. An appropriate title for an Active Directory schema self-help book would be *Everything I Need to Know I Learned from Myself.*

The Active Directory schema contains two schema object classes, ClassSchema and AttributeSchema. Objects derived from these classes act like patterns in a lathe to turn out other objects. The schema objects are stored in the directory in the `cn=Schema,cn=Configuration,dc=<domain_name>,dc=<domain_root>` container.

In addition to ClassSchema and ClassAttribute classes, the Schema container holds a class called SubSchema with one instance, an object called Aggregate. The distinguished name of this object is `cn=aggregate,cn=schema,cn=configuration,dc=company,dc=com`. The purpose of Aggregate is to provide a single point for LDAP clients to discover information about the Active Directory schema. Without this object, clients would be forced to perform expensive scans of the entire Schema container.

Identifying Objects

We've completed the overview of the schema structure, function, and rules. Before moving forward, let's look at how Active Directory uniquely identifies objects. This information is crucial to understanding the more advanced Active Directory tools. Here is a brief attribute listing for a sample User object made using the LDIFDE utility. The unique identifiers are highlighted:

```
C:\>ldifde -d cn=bgates,cn=users,dc=dotnet,dc=com -f con
Connecting to "DC01.Company.com"
Logging in as current user using SSPI
Exporting directory to file con
Searching for entries...
Writing out entries.dn: CN=bgates,CN=Users,DC=dotnet,DC=com
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: bgates
distinguishedName: CN=bgates,CN=Users,DC=dotnet,DC=com
instanceType: 4
whenCreated: 20020812134034.0Z
```

```
whenChanged: 20020812134034.0Z
uSNCreated: 13772
uSNChanged: 13774
name: bgates
objectGUID:: 7swJ8PXwqkWu8N2Qv+jQ+Q==
userAccountControl: 512
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 0
lastLogoff: 0
lastLogon: 0
pwdLastSet: 126736332347481024
primaryGroupID: 513
objectSid:: AQUAAAAAAAUVAAAAdbl1VBUlr0cWwOoyVQQAAA==
accountExpires: 0
logonCount: 0
sAMAccountName: bgates
userPrincipalName: bgates@dotnet.com
sAMAccountType: 805306368
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=dotnet,DC=com
```

Distinguished Name

Because LDAP uses an object-oriented database, it is important that each object has a unique path in the namespace, similar to the way that a filename and path must be unique in a file system.

The Distinguished Name (DN) attribute of an object defines the LDAP path all the way to the root of the namespace; therefore, the DN must be unique. If you move an object to a different container in Active Directory, in reality, you are simply changing the DN.

Globally Unique Identifier (GUID)

In classic Exchange, Microsoft used the DN as the unique database row identifier for objects in the directory service store. This unfortunate engineering decision created a configuration problem for Exchange. When an object is moved, its DN changes, but a unique row identifier in a database cannot ever change. For this reason, in Exchange 5.5 and earlier, mailbox recipients cannot be moved but must be freshly created and then linked to a User account in the SAM.

To avoid that problem in Active Directory, Microsoft used a different unique row identifier called the *Globally Unique Identifier*, or GUID . A GUID is created using an algorithm that virtually guarantees its uniqueness within a system.

Using a GUID permits you to move objects at will between containers in Active Directory without changing the unique row numbers for the objects, thereby maintaining internal referential integrity in the database. Keep this behavior in mind, because you'll see it at work when we discuss the role of the Infrastructure Master in keeping track of group members from other domains.

Other Uses for GUIDs

Microsoft uses the GUID algorithm in a variety of different circumstances. You will see them in designators used to identify COM objects and OLE registrations. Group policies use the GUID algorithm to create a unique folder name for each policy. The operating system identifies hardware using GUIDs during Plug-and-Play enumeration. GUIDs also go by the names *Universally Unique Identifier* (UUID) and *Class ID* (CLSID).

Security Identifier (SID)

Three classes of Active Director objects can be placed on the access control lists (ACLs) used to protect security objects. These object classes are User, Computer, and Group. Together, they are termed *security principals*.

A security principal is assigned a unique number called a *Security Identifier*, or SID. This is exactly the same SID used by NT to identify users, groups, and computers. A SID for a security principal is made up of the SID of the security principal's domain and a unique suffix, called a Relative ID, or RID. The series of RIDs for security principals that can be created by an administrator start at decimal 1000. For example, the first User account created following the creation of a domain would be given RID 1000. The next object, call it a group, would be RID 1001, and so forth.

The combination of a domain SID and a RID form a unique number within a domain and within a forest. The pool of RIDs is maintained by a specially designated Windows Server 2003 domain controller called a RID Master.

SAM Account Name

In an NT domain, every object in the SAM must have a unique name. This is true for computers, users, and groups. A unique name guarantees that the object will have a unique NetBIOS presence in the network as well as a one-to-one correspondence between the logon name (in the case of users and computers) and the SID used to control resource access.

The same restriction is left in place in Windows 2000 and Windows Server 2003. Every user, computer, and group in a domain must have a unique name. This attribute is called `SAMAccountName`, although you might hear it called *logon name* or *flat name*. When you create a new security principal, regardless of the container where you place the object, it must have a unique flat name in the domain.

User Principal Name (UPN) and Service Principal Name (SPN)

Just as unique flat names identify security principals in NetBIOS, User Principal Names (UPNs) identify security principals within the hierarchical LDAP namespace in Active Directory. A UPN takes the form `User@Company.com`.

Unique UPNs ensure that users can log on with their UPN rather than the classic *`domain\username`* construct. The Global Catalog is used to "crack" the UPN into its constituent parts.

To assure uniqueness, when a security principal is created, the system refers to the Global Catalog to verify that the UPN has not already been used. If a GC server is not available, the system displays an error message prompting the administrator to wait until a GC is available so that uniqueness can be verified.

In a Parent/Child trust configuration, the UPN suffix of the root domain is assigned to every security principal. In a Tree Root trust configuration, you must manually assign a common UPN suffix. This is done using the Properties window of the domain tree in the AD Domains and Trusts console.

Object Identifier (OID)

In addition to the attributes that assure uniqueness of a particular object, Active Directory needs a way to assure that objects of the same class all come from the same Schema object. This is done by assigning a unique Object Identifier, or *Object Identifier* (OID) to each object in the Schema naming context. ISO defines the structure and distribution of OIDs in `ISO/IEC 8824:1990`, "Information Technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)."

ASN.1 provides a mechanism for standards bodies in various countries to enumerate standard data items so that they do not conflict with one other. ASN.1 governs more than just directory services classes and attributes. For example, OIDs are used extensively in SNMP to build hierarchies of Management Information Base (MIB) numbers. They are also assigned to many items associated with the Internet. If you're interested in the list of organizations that assign OID numbers and their hierarchy, it is available at `ftp.isi.edu/in-notes/iana/assignments/enterprise-numbers`.

If you ever need to create a new attribute or object class in Active Directory, you must have a unique OID. There are a couple of ways to get one. The first is to apply to ANSI for your own numerical series. This costs a few thousand dollars and takes a while to process. The other is to use the OIDGEN utility from the Resource Kit. This will generate a Class and an Attribute OID out of Microsoft's address space. The disadvantage to using OIDGEN is that the resultant number is very, very, very long. Here is an example:

```
C:\>oidgen
Attribute Base OID:
1.2.840.113556.1.4.7000.233.180672.443844.62.26102.2020485.1873967.207938
    Class Base OID:
1.2.840.113556.1.5.7000.111.180672.443844.62.199519.642990.1996505.1182366
```

Finding OID Hierarchy Information

Many thanks to Harald Alvestrand, who made good use of a long winter in Trondheim, Norway, to build a hyperlinked tree showing many of the common OID registrations. His information is now slightly out of date but the structure is still valid and very instructive. Visit his web site at `www.alvestrand.no/objectid`.

Active Directory Support Files

The ESE engine used by Active Directory is based on Microsoft's Jet database technology. Jet uses a b-tree file structure with transaction logs to ensure recoverability in the event of a system or drive failure.

When you promote a server to a domain controller, you select where to put the Active Directory files. The default path is in the boot partition under `\Windows\NTDS`. Generally, it is a good idea to put them on a separate volume from the operating system files to improve performance.

The following list contains the Active Directory support files and their functions:

- **Ntds.dit.** This is the main AD database. NTDS stands for *NT Directory Services*. The DIT stands for *Directory Information Tree*. The Ntds.dit file on a particular domain controller contains all naming contexts hosted by that domain controller, including the Configuration and Schema naming contexts. A Global Catalog server stores the partial naming context replicas in the Ntds.dit right along with the full Domain naming context for its domain.
- **Edb.log.** This is a transaction log. Any changes made to objects in Active Directory are first saved to a transaction log. During lulls in CPU activity, the database engine commits the transactions into the main Ntds.dit database. This ensures that the database can be recovered in the event of a system crash. Entries that have not been committed to Ntds.dit are kept in memory to improve performance. Transaction log files used by the ESE engine are always 10MB.
- **Edbxxxxx.log.** These are auxiliary transaction logs used to store changes if the main Edb.log file gets full before it can be flushed to Ntds.dit. The xxxxx stands for a sequential number in hex. When the Edb.log file fills up, an Edbtemp.log file is opened. The original Edb.log file is renamed to Edb00001.log, and Edbtemp.log is renamed to Edb.log file, and the process starts over again. ESENT uses circular logging. Excess log files are deleted after they have been committed. You may see more than one Edbxxxxx.log file if a busy domain controller has many updates pending.
- **Edb.chk.** This is a *checkpoint file*. It is used by the transaction logging system to mark the point at which updates are transferred from the log files to Ntds.dit. As transactions are committed, the checkpoint moves forward in the Edb.chk file. If the system terminates abnormally, the pointer tells the system how far along a given set of commits had progressed before the termination.
- **Res1.log** and **Res2.log.** These are reserve log files. If the hard drive fills to capacity just as the system is attempting to create an Edbxxxxx.log file, the space reserved by the Res log files is used. The system then puts a dire warning on the screen prompting you to take action to free up disk space quickly before Active Directory gets corrupted. You should never let a volume containing Active Directory files get even close to being full. File fragmentation is a big

performance thief, and fragmentation increases exponentially as free space diminishes. Also, you may run into problems as you run out of drive space with online database defragmentation (compaction). This can cause Active Directory to stop working if the indexes cannot be rebuilt.

- **Temp.edb.** This is a scratch pad used to store information about in-progress transactions and to hold pages pulled out of Ntds.dit during compaction.
- **Schema.ini.** This file is used to initialize the Ntds.dit during the initial promotion of a domain controller. It is not used after that has been accomplished.

Active Directory Utilities

We've now seen all the components in Active Directory. Over the next few chapters, we'll see how to use those components to build a reliable, useful structure. First, though, let's take a look at the tools of the trade for Active Directory. You get some of these tools when you promote Windows Server 2003 to a domain controller. Others come from the support tools on the Windows Server 2003 CD. Others require purchasing the Resource Kit. I'll identify the origin as I discuss the tools.

Standard Active Directory Management Consoles

Windows Server 2003 comes with three standard MMC-based consoles for viewing and managing Active Directory objects. MMC console files have an .msc extension. The management consoles can be differentiated by the naming context they are used to manage:

- **AD Users and Computers.** This console is used to manage the contents of a Domain naming context. The console name is Dsa.msc.
- **AD Sites and Services.** This console is used to manage the Sites and Services containers inside the Configuration naming context. The console filename is Dssite.msc.
- **AD Domains and Trusts.** This console is used to manage the contents of the Partitions container inside the Configuration naming context. It uses the CrossRef objects in the Partitions container to identify domains in the forest in their assigned hierarchy. The console filename is Domain.msc.

These consoles can all be launched from the Start button at Windows Server 2003 using START | PROGRAMS | ADMINISTRATIVE TOOLS | <CONSOLE NAME>. You can also launch them by entering the name of the MMC console file, such as Dssite.msc, in a Run window or on the command line. Specific instructions for using these AD management consoles are contained in the remaining Active Directory chapters. The most important thing to note at this time, as you get familiar with them, is that virtually all functionality is available from a right-click of the mouse. Very few features require operations from the menu.

Virtual List Views

If you have experience with Windows 2000, you may notice a difference in the way Windows Server 2003 displays pick lists that are built as a result of LDAP searches.

In Windows 2000, the results of the search were delivered to the client in increments of 1500 ordered sequentially as matches were found in the directory. This made pick lists difficult to manage because the items were not sorted.

In Windows Server 2003, search results are fully collected and sorted at the server then delivered in increments stipulated by the client. This means that pick lists are automatically sorted alphabetically, making it easier to locate a particular item.

Schema Console

Microsoft makes it fairly difficult to get access to the Schema naming context. It does not include a standard MMC console for managing the schema. You must create a custom console that contains the Schema snap-in. A snap-in is a *Dynamic Link Library* (DLL) that is loaded by the MMC executable. After you have associated one or more snap-ins with a console, you can save the console with a unique name that has an .msc extension.

Before you can create a custom MMC console for schema management, you must have access to the Schema snap-in. This snap-in is part of the administrative tools but is not registered by default. This prevents casual monkeying around with the schema. To register the Schema snap-in, open a command console, navigate to `C:\Windows\System32`, and run `regsvr32 schmmgmt.dll`.

After the Schema snap-in is registered, create a custom MMC console for it as directed in Procedure 6.1.

Procedure 6.1 Creating a Custom Schema Management Console

1. From the Run window, type **`mmc`** and click `OK`. This opens an empty MMC console.
2. From the CONSOLE menu, select FILE | ADD/REMOVE SNAP-IN. The Add/Remove Snap-in window opens.
3. Click `Add`. The Add Standalone Snap-in window opens.
4. Double-click `Active Directory Schema` and then click `Close`.
5. Click `OK` to save the change and return to the MMC window. The `Active Directory Schema` tree will appear under the `Console Root` folder.
6. Save the file with a name like Schema.msc. The system will put the file in your personal profile. Save it to the \Windows\System32 folder if you want other administrators to use it.

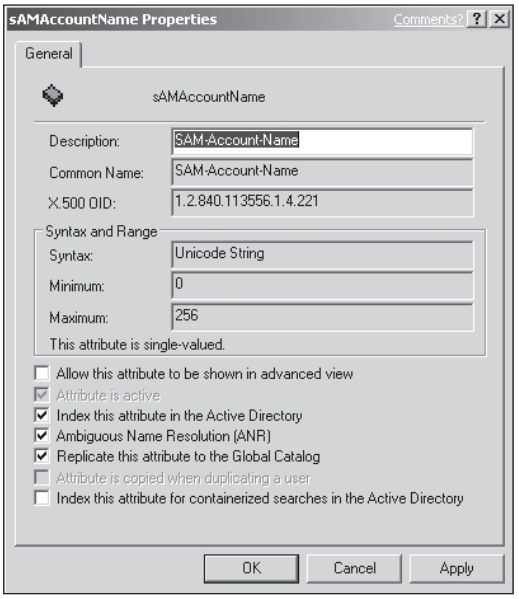

Figure 6.16 Properties window for the `attributeSchema` object used to create the `SamAccountName` attribute.

When you expand the Schema tree, you'll see the objects that make up the classes and attributes of the schema. You can double-click to see the properties for one of these objects. Figure 6.16 shows an example of the properties for the SamAccountName attribute, which holds a user's logon name.

There is a list of options that affect how this attribute will be used in Active Directory:

- **`Allow This Attribute To Be Shown In Advanced View`**. Each AD console has an ADVANCED VIEW option. This prevents cluttering the interface with options that are only used occasionally. (It also confounds administrators who are trying to perform an operation and don't know that the option is hidden in a normal view. This is called a *feature*.)
- **`Attribute Is Active`**. Some attributes are not required for system operation and can be disabled to prevent them from getting values.
- **`Index This Attribute In Active Directory`**. Like any database, performance improves when you search for indexed attributes. Indexing consumes disk space and processor time, though, and an attribute must be unique to make the index worthwhile. Only the most commonly searched attributes are selected for indexing.

- **Ambiguous Name Resolution (ANR).** ANR permits searching for partial matches. An ANR search for a SamAccountName of "gh" would return "ghawn," "ghaskell," "ghowell," and so forth. ANR searches put quite a strain on the database engine, so only nine attributes are selected to use it by default. If you design an application with an attribute that would benefit from ANR searching, you can use this option to add it to the ANR set.
- **Replicate This Attribute To The Global Catalog.** This setting determines if an attribute should be included in the Global Catalog. Only commonly searched attributes are included to minimize GC size and replication load. In Windows 2000, adding or removing an attribute from the GC required a full GC rebuild and replication. This had the potential for creating significant traffic. Windows Server 2003 permits modifying the contents of the GC without forcing a full rebuild.
- **Attribute Is Copied When Duplicating A User.** With this option selected, the value for the attribute would be carried over to a new User object with the Copy function. The SamAccountName must be unique in a domain, so this option is disabled for this attribute.
- **Index This Attribute For Containerized Searches In The Active Directory.** The search routines provided in the LDAP API and with Microsoft ADSI permits searching a container rather than the entire directory. You can select this option to improve lookup times for container searches.

The schema can only be modified at one domain controller, the one designated as a *Schema Operations Master*. This ensures the integrity of the schema by preventing potentially conflicting changes from being made at two different domain controllers during the same replication interval.

You can identify the Schema Operations Master by right-clicking Active Directory Schema and selecting OPERATIONS MASTER from the flyout menu.

You do not need to be at the console of the Schema Operations Master server to view and modify the schema. You can put the focus of the Schema console on this server by right-clicking Active Directory Schema and selecting CHANGE DOMAIN CONTROLLER.

Search Flags

Several of the attribute property options listed in the Schema Manager control a value called SearchFlags. This value controls the following actions (values are additive):

1 = Index this attribute

2 = Index this attribute and its container

4 = Add to ANR set (must have indexing set)

8 = Keep the attribute when deleting the object and creating a tombstone

16 = Copy the attribute's value when creating a new copy of an object

Of these settings, only number "8" cannot be controlled from the Schema Manager snap-in. You can use the ADSI Edit console (covered in the next section) to change the value.

Registry Requirements for Schema Modifications
Windows 2000 had a security measure that required a special `Schema Updated Allowed` parameter in the Registry of the machine where you ran the Schema console. This requirement has been removed in Windows Server 2003.

You must be a member of the Schema Admins group to modify any part of the schema. By default, the Administrator account is a member of this group. The Schema Admins group has a set of special permissions for the Schema container. These include the following:

- Change Schema Master
- Manage Replication Topology
- Replicating Directory Changes
- Replication Synchronization
- Update Schema Cache

You should not make changes to the schema unless you are very familiar with its structure and what you want to accomplish. New schema objects cannot be deleted. Changes to existing objects can cause problems that could force reinstalling Active Directory from scratch or recovering from a backup tape.

General-Purpose Active Directory Tools

The standard AD management consoles provide a feature-rich interface for accessing and modifying Active Directory objects and attributes. They also hide a lot of the gears and pulleys that go together to make Active Directory work.

We're now going to take a look at a few tools that take us behind the glitzy façade of those fancy AD management consoles. We're going to see the real world that underlies Active Directory. If you've ever seen *The Matrix*, you have an idea of what we're in for. I have just one question before we start:

Do you want to take the red pill or the blue pill?

ADSI Edit

The first set of general-purpose tools we'll look at come in the suite of Support Tools on the Windows Server 2003 CD. Install the support tools by double-clicking the `\Support\Tools\2000RKST.MSI` icon and walking through the installation wizard.

After the tools are installed, open a Run window and enter **`adsiedit.msc`**. This is the console filename for the ADSI Editor. When the ADSI Edit console opens, you see icons representing the three standard naming contexts for a domain controller: `Domain NC`, `Configuration Container`, and `Schema`. (It cannot display the Application naming contexts.) See Figure 6.17 for an example.

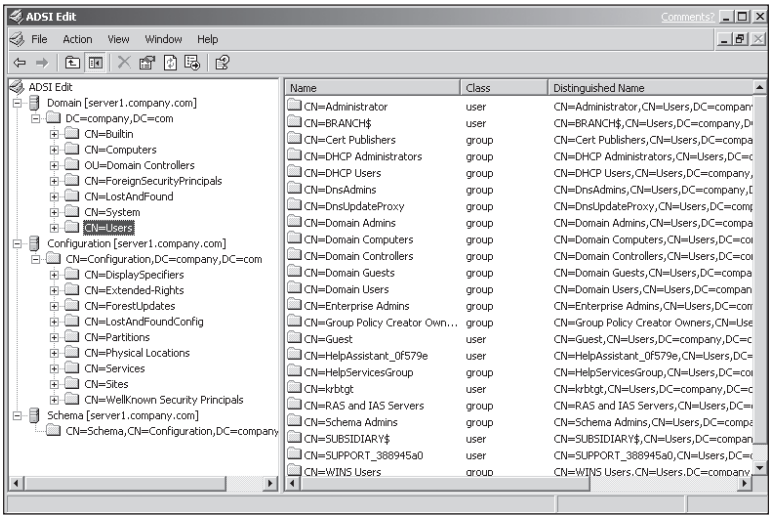

Figure 6.17 ADSI Edit console showing the three standard naming contexts for a domain controller.

Selecting Alternative Domain Controllers for ADSI Edit

If you do not see any naming contexts when you open ADSI Edit, or you want to view a naming context on another domain controller, proceed as follows:

1. Right-click the `ADSI Edit` icon and select CONNECT TO from the flyout menu. The Connection window opens.
2. Under `Computer`, select the `Select or Type a Domain or Server` radio button.
3. In the combo box, type the fully qualified DNS name of a domain controller. When you make this entry, the `Path` entry automatically changes.
4. Click `Advanced`. The Advanced window opens (see Figure 6.18).

The options in this window are used as follows:

- **Credentials.** If you are connecting to Active Directory in another domain, or you are currently logged on using an account that does not have administrator privileges, you can specify a set of administrator credentials.
- **Port Number.** If this field is left blank, ADSI Editor uses well-known TCP port 389 for LDAP. You can specify a different port if you are browsing a non-standard implementation. You could also use this option to browse the Global Catalog through TCP port 3268, but it is more convenient to use the Protocol feature.
- **Protocol.** Select whether you want to browse Active Directory (port 389) or the Global Catalog (port 3268).

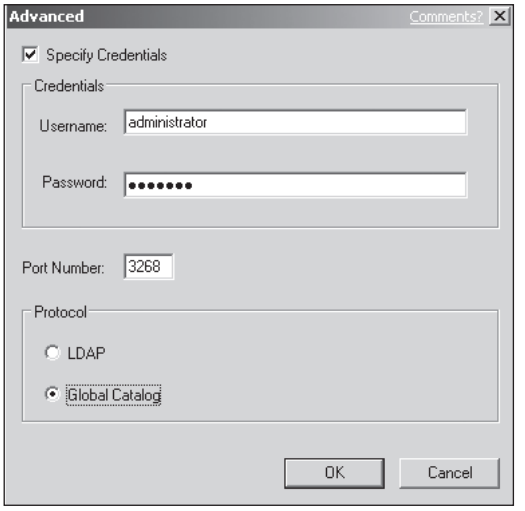

Figure 6.18 ADSI Editor Advanced window showing alternative credentials, specific port number, and protocol selection.

5. Click OK to save the changes and return to the Connections window.
6. Click OK to save the changes and return to the main ADSI Edit console. The display refreshes to show the new settings, if you made any changes.

Using ADSI Edit to View and Modify AD Objects

Use the steps in Procedure 6.2 to view and modify information about Active Directory objects.

Procedure 6.2 Using the ADSI Editor to View AD Objects

1. Expand the tree to show the top of the naming context you want to view. You can open several Domain naming contexts from several domain controllers at the same time, making ADSI Edit a handy way to view a big enterprise.
2. You can view the attributes associated with any object in any naming context. For example, expand the Domain NC tree to show the list of objects under `cn=Users` and then right-click `cn=Administrator` and select PROPERTIES from the flyout menu. The Properties window opens (see Figure 6.19).

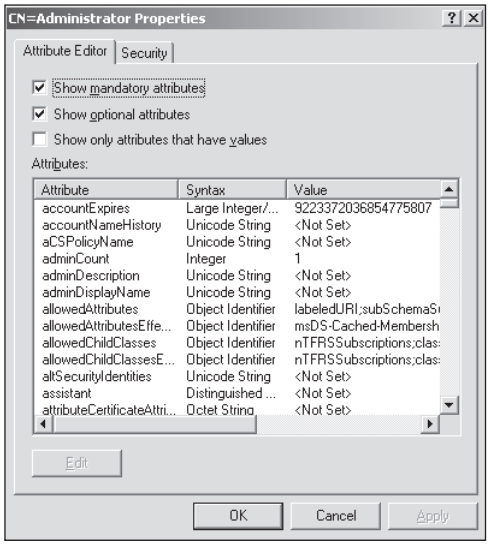

Figure 6.19 Properties for distinguished name `cn=Administrator,cn=Users, dc=Company,dc=com`.

3. The `Show Mandatory Attributes` and `Show Optional Attributes` options are checked by default. Select the `Show Only Attributes That Have Values` option to eliminate extraneous information in the window.
4. Scroll down through the window to view the various attributes and their values.
5. To change a value, double-click it. ADSI Edit will select the appropriate low-level editor to modify the attribute.

Think of ADSI Edit as a kind of super Regedit for Active Directory. All the same caveats apply. You can turn a perfectly tuned domain into sad, twisted carnage with a few mouse clicks. You can also perform miraculous surgery that solves seemingly intractable problems.

LDAP Browser

ADSI Edit is built from the ground up as a tool to manage Active Directory naming contexts. The Support Tools also includes a generic LDAP tool that is capable of accessing any RFC-compliant LDAP directory service. This tool is a true executable, not an MMC snap-in. It is called the LDAP Browser, or Ldp.exe.

LDP is a little less convenient to use than ADSI Edit, and it requires you to know a little more about how to use LDAP. But it's well worth the effort to learn. LDP provides a lot more information with a single mouse click than ADSI Edit. Also, some LDAP operations are hidden by ADSI Editor but exposed by LDP.

Installing LDP

When you use LDP, you must walk through a few steps to bind (authenticate) and set up to view the directory tree. Procedure 6.3 demonstrates how it works.

Procedure 6.3 Binding with LDP

1. At a client in a domain, open the Run window and enter `LDP`. This opens the LDP window.
2. Select CONNECTION | BIND to open the Bind window.
3. Enter administrator credentials in the domain or forest.
4. Click `OK`. The attributes associated with the RootDSE object appear in the right pane. These attributes show the structure and content of the directory on the server. (LDP will bind to your logon server. You can use the `Connect` option to select another server.)
5. From the menu, select VIEW | TREE. This opens the Tree View window.
6. Under `BaseDN`, enter the distinguished name of the container you want to browse. For example, you can enter `dc=Company,dc=com` to start at the top of the Domain naming context for the `Company` domain. You can also specify a container lower in Active Directory. For example, you could select the Users container by entering `cn=Users,dc=Company,dc=com`. The interface is not case sensitive.
7. Click `OK`. The left pane of the window now shows the root of the container you entered. Click the + sign or double-click the name to expand the tree. This generates an LDAP query that enumerates the child objects in the container, which are listed in the tree in the left pane. It also generates a query for the attributes associated with the domain object. These are listed in the right pane (see Figure 6.20).

Searching for a Specific Attribute

LDP is also a convenient place to search the directory for specific instances of an attribute (see Procedure 6.4).

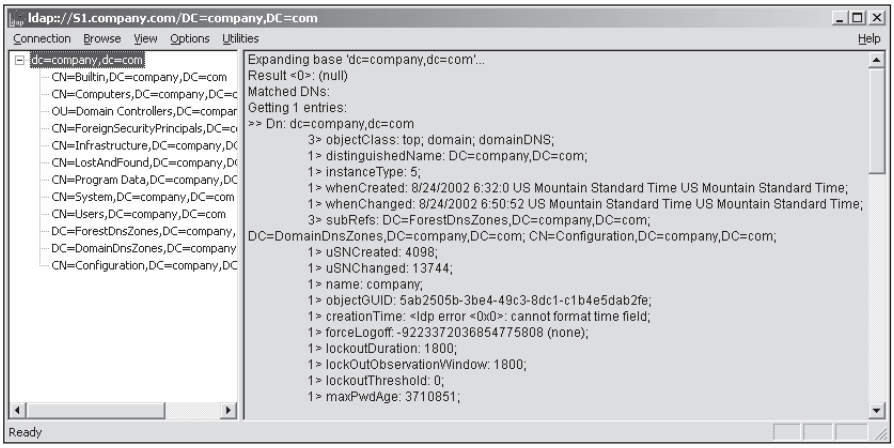

Figure 6.20 LDP window showing tree view of `Company.com` domain.

Procedure 6.4 Searching with LDP

1. Select BROWSE | SEARCH from the main menu to open a search window.
2. In `Base DN`, enter the distinguished name of the container you want to search. You can enter the DN of the root domain of a tree if you want to search the entire tree, but this might take a while if you have a large enterprise with several child domains.
3. In `Filter`, enter the search criteria. The syntax is a little tricky. LDAP expects to see Boolean operators such as `&` (AND) and `|` (OR) at the beginning of the search string. For example, if you want to find all Users who are in the Finance department, you would enter `(&(objectclass=user)(department=finance))`. The entry is not case-sensitive.
4. If you want to search just the object you entered the DN for, select Base. If you want to search the base object and any objects directly under it, select `One Level`. If you want to search all containers under the base container, select `Subtree`. LDP cannot search an entire forest. You must select a base DN at the root of each tree in the forest and run the search multiple times.

LDP Search Wildcards

LDP only accepts wildcards at the middle and end of a filter option. You can search for `department=fin*` or for `department=fin*ce` but not for `department=*ance`.

You can do many fancy tricks with LDP. You can get a quick view of the security descriptor for an object. You can view the replication metadata associated with all the properties of an object (something we'll cover in more detail in Chapter 7, "Managing Active Directory Replication"). It's well worth your time to learn the ins and outs of LDP. You'll also learn a lot about LDAP and Active Directory at the same time. You'll be glad you took the red pill.

DCDIAG

This tool comes in the Resource Kit. It is an invaluable diagnostic utility for examining and troubleshooting a variety of Active Directory operations. You'll find that these tests give great information about the current state of your Active Directory domains, trusts, and replication status. Enter **`dcdiag /?`** to get a list of the tests that are performed. Every element of Active Directory operation is tested. This utility is highly recommended.

DS Tools

Windows Server 2003 expands the number of command-line tools available for administering Active Directory with a set of DS tools. Here is a list:

- **Dsadd.** Creates an object of a specified class. A wide variety of attributes can be given values at the same time. For example, here are the attributes for `dsadd user`:

```
dsadd user <UserDN> [-samid <SAMName>] [-upn <UPN>] [-fn <FirstName>]
  [-mi <Initial>] [-ln <LastName>] [-display <DisplayName>]
  [-empid <EmployeeID>] [-pwd {<Password> | *}] [-desc <Description>]
  [-memberof <Group ...>] [-office <Office>] [-tel <Phone#>]
  [-email <Email>] [-hometel <HomePhone#>] [-pager <Pager#>]
  [-mobile <CellPhone#>] [-fax <Fax#>] [-iptel <IPPhone#>]
  [-webpg <WebPage>] [-title <Title>] [-dept <Department>]
  [-company <Company>] [-mgr <Manager>] [-hmdir <HomeDir>]
  [-hmdrv <DriveLtr:>] [-profile <ProfilePath>] [-loscr <ScriptPath>]
  [-mustchpwd {yes | no}] [-canchpwd {yes | no}]
  [-reversiblepwd {yes | no}] [-pwdneverexpires {yes | no}]
  [-acctexpires <NumDays>] [-disabled {yes | no}]
  [{-s <Server> | -d <Domain>}] [-u <UserName>]
   [-p {<Password> | *}] [-q] [{-uc | -uco | -uci}]
```

- **Dsmod.** Modifies selected attributes of an existing object.
- **Dsrm.** Removes an object or container. Use caution. You can accidentally remove an entire branch of the tree and force yourself into a tape restore to recover.
- **Dsmove.** Moves an object to a new container. The container must be in the same naming context.
- **Dsquery.** Finds objects that match a specified search criteria.
- **Dsget.** Views selected properties from a specified object.

Bulk Imports and Exports

You may find yourself in a situation where you want to dump information out of Active Directory into a flat file for searching. Or you may need to create large numbers of objects and you want to simplify your work by importing information from a flat file. A standard Windows domain controller has a couple of utilities that help with this kind of bulk object processing. First, we need to take a look at the format for exchanging LDAP information.

LDAP Data Interchange Format (LDIF)

RFC 2849, "The LDAP Data Interchange Format (LDIF)—Technical Specification" defines a standard structure for exchanging LDAP information. The following example shows the LDIF format for the attributes of the Administrator account in the `Company.com` domain:

```
dn: CN=Administrator,CN=Users,DC=company,DC=com
memberOf: CN=Group Policy Admins,CN=Users,DC=company,DC=com
memberOf: CN=Enterprise Admins,CN=Users,DC=company,DC=com
memberOf: CN=Schema Admins,CN=Users,DC=company,DC=com
memberOf: CN=Administrators,CN=Builtin,DC=company,DC=com
memberOf: CN=Domain Admins,CN=Users,DC=company,DC=com
accountExpires: 9223372036854775807
adminCount: 1
badPasswordTime: 125693193676075896
badPwdCount: 0
codePage: 0
cn: Administrator
countryCode: 0
description: Built-in account for administering the computer/domain
isCriticalSystemObject: TRUE
lastLogoff: 0
lastLogon: 125693891796993128
logonCount: 109
distinguishedName: CN=Administrator,CN=Users,DC=company,DC=com
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=company,DC=com
objectClass: user
objectGUID:: gLgtb/ju0hGcKADAT1NqTQ==
objectSid:: AQUAAAAAAAUVAAAAoF4uDLI/DAf7Cwgn9AEAAA==
primaryGroupID: 513
pwdLastSet: 125681556744344992
name: Administrator
sAMAccountName: Administrator
sAMAccountType: 805306368
userAccountControl: 66048
uSNChanged: 1532
uSNCreated: 1410
whenChanged: 19990410040835.0Z
whenCreated: 19990410034956.0Z
```

There are a few items of note with this example:

- LDIF files use simple ASCII characters. If you have high-order Unicode values in some of the attributes, they might not survive the translation.
- Long integers that represent time and dates will be represented in decimal format and, as such, will not survive reimport. These items are discarded and created afresh when an entry is imported and a new object created.
- Octet strings are converted to Base64 format. This is indicated by a double-colon after the attribute name. *ObjectGUID* is an example. These values withstand a reimport. For the most part, though, this syntax is used for values that are unique for an object so the imported values would be ignored.
- The attributes conform to the Active Directory schema for the forest where they were obtained. Attempting to import these values into a foreign directory service can result in compatibility issues. At the very least, you'll need to change the distinguished names, because it is unlikely that the foreign directory service would use the same namespace.

The LDIF standard includes several command verbs that are used to determine what to do with a particular record. These verbs permit adding, modifying, replacing, or deleting an entire object or individual attributes of an object. They also permit modifying the directory schema. Active Directory permits LDIF to add and modify object classes and attributes, but it does not permit them be deleted. After a class or attribute has been added to the schema, it's there to stay.

LDIF and Active Directory Schema Upgrades

Lest you think that LDIF is one of those obscure programmer toys that reasonable system administrators should avoid like it was oozing with plague, consider this: When you upgrade the first Windows 2000 domain controller in a domain to Windows Server 2003, new objects are added and old objects modified to support changes in the new operating system version. In addition, the Active Directory schema must be modified to support the new features in Windows Server 2003. How does Microsoft install these schema updates? With LDIF files, that's how.

Check the Windows Server 2003 CD in the `\I386` folder. Look for a series of files with an LDF extension. These contain the LDIF entries that modify Active Directory contents and the schema. The CD includes an uncompressed executable called Schupgr.exe. This executable loads the changes from the LDF files into Active Directory.

One last feature of this upgrade method is important to note. The last step in each LDF file modifies an attribute of the Schema container called ObjectVersion. This is how Windows keeps track of the LDF files applied by Windows updates. Installing Windows Server 2003 upgrades the schema to version number 30. Installing Exchange also modifies the schema but does not change the schema version number.

LDIFDE

A Windows domain controller comes with a command-line tool for importing and exporting LDIF files, LDIFDE. Run **`ldifde`** with no switches to get a list of parameters.

LDIFDE simplifies importing and exporting large numbers of records to and from Active Directory, but it also comes in handy for making quick checks of directory entries without opening up a pesky MMC snap-in. Use the `-f con` switch to direct the output to the console. For example:

- To know the group membership of a user, use `Ldifde -d cn=username,cn=Users, dc=company,dc=com -f con`.
- To check the entries in a trusted domain, use `Ldifde -s alb-dc-01.office. company.com -d dc=Office,dc=Company,dc=com -f con`.
- To find all the printers in an organizational unit, use `Ldifde -d ou=Phoenix, dc=Company,dc=com -r "(objectclass=printers)" -f con`.

You can use LDIFDE to dump a file of information about a user and then modify the settings and the username and import that file as a new user. To do this, use the `-m` option to remove the SAM-specific information from the dump file.

You can also use LDIFDE to modify attributes of existing objects, but you need to know a little trick. After you've created an LDIF file consisting of entries you want to modify, you must put a dash on the first blank line at the end of the entries and then a blank line after that. Here's an example showing how to change the `Description` attribute for a user named Avguser:

```
dn: CN=avguser,OU=Phoenix,DC=company,DC=com
changetype: modify
replace: Description
Description: Wazula
-
```

Without that dash, you'll get an error similar to the following:

```
Failed on line 4.  The last token starts with 'W'. The change-modify entry is
missing the terminator '-'.
```

CSVDE

Working with the LDIF format can get a little tedious because it sorts attributes vertically rather than horizontally. If you prefer a more standard spreadsheet layout, use the CSVDE utility. The switches for CSVDE are the same as for LDIFDE.

Here's an example of using CSVDE. Let's say you are the administrator for a school district and you want to add 5000 new students into Active Directory. Your student list may be in a mainframe or AS400 application or a UNIX application of one form or another or a SQL database. You can write a little JCL (Job Control Language) routine or do a quick SQL query to output the student list to a delimited file. Import the delimited file into a spreadsheet and massage it until you get the required data for Active Directory. (Do a `csvde -f output.ldf` to see the column headings and data types.) Then use `csvde -i` to import the spreadsheet contents into Active Directory.

Reimporting LDIF Dumps

If you do an LDIFDE or CSVDE export, many of the attributes for user and group objects are owned by the system and cannot be reimported. Here's a trick. Run the export with the `-m` switch. This enables SAM Logic, which is another way of saying that the export skips the attributes that are owned by the system. This gives you a template to use when building your import files or spreadsheets.

Other LDAP Tools

Because Active Directory is an RFC-compliant implementation of LDAP, you can use virtually any LDAP tool for browsing objects and collecting information. Here are a few sources of LDAP tools and related information:

- **OpenLDAP** (`www.openldap.org`). If you are an open source kind of person, you should take a look at the latest wares from that community. These toolkits are not for the fainthearted, and there are no compiled packages to play with, but it's worth a peek if you want to build your own administration tools to replace those clumsy MMC snap-ins.
- **Novell** (`www.novell.com/products/nds/ldap.html`). NetWare 5 boogies on IP and so does NDS. Novell is putting lots of calories into doing the "Internet thing" right. Also take a look at `developer.novell.com` for LDAP and X.500 tools that might be useful in a mixed network.

Moving Forward

This chapter covered the structure and operation of Active Directory. The next five chapters describe how to design, deploy, and manage Active Directory-based domains, how to manage replication between domain controllers, and how to repair and recover Active Directory in the event of a problem.