- Table of Contents
- Index
- Reviews
- Reader Reviews
- Errata

Active Directory, 2nd Edition

By Robbie Allen, Alistair G. Lowe-Norris

Publisher: O'Reilly
Pub Date: April 2003
ISBN: 0-596-00466-4
Pages: 686

Active Directory, 2nd Edition, provides system and network administrators, IT professionals, technical project managers, and programmers with a clear, detailed look at Active Directory for both Windows 2000 and Windows Server 2003. Active Directory, 2nd Edition will guide you through the maze of concepts, design issues and scripting options enabling you to get the most out of your deployment.

Active Directory, 2nd Edition

By Robbie Allen, Alistair G. Lowe-Norris

Start Reading

Publisher: O'Reilly
Pub Date: April 2003
ISBN: 0-596-00466-4
Pages: 686

Copyright

Printed in the United States of America.

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

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

Preface

Active Directoy is a common repository for information about objects that reside on the network, such as users and groups, computers and printers, and applications and files. The default Active Directory schema supports numerous attributes for each object class that can be used to store a variety of information. Access Control Lists (ACLs) are also stored with objects, which allow you to maintain permissions for who can access and manage them. Having a single source for this information makes it more accessible and easier to manage. However, to accomplish this with Active Directory requires a significant amount of knowledge of such topics as LDAP, Kerberos, DNS, multi-master replication, group policies, and data partitioning, to name a few. This book will be your guide through this maze of technologies, showing you how to deploy a scalable and reliable Active Directory infrastructure.

Windows 2000 Active Directory has proven itself to be very solid in terms of features and reliability, but after several years of real-world deployments, there was much room for improvement. With Windows Server 2003, Microsoft focused on security, manageability, and scalability enhancements that are sure to make even the most recent Windows 2000 deployers consider upgrading. Fortunately, Microsoft has made the upgrade process to Windows Server 2003 Active Directory seamless. You can proceed at your own pace based on how quickly you need to upgrade.

This book is a significant update to the very successful first edition. All of the existing chapters have been brought up to date with Windows Server 2003, and eight additional chapters have been included to explain new features or concepts not covered in the first edition. This second edition describes Active Directory in depth, but not in the traditional way of going through the graphical user interface screen by screen. Instead, the book sets out to tell administrators exactly how to design, manage, and maintain a small, medium, or enterprise Active Directory infrastructure. To this end, the book is split up into three parts.

Part I introduces in general terms much of how Active Directory works, giving you a thorough grounding in its concepts. Some of the topics include Active Directory replication, the schema, application partitions, group policies, and interaction with DNS.

In Part II we describe in copious detail the issues around properly designing the directory infrastructure. Topics include in-depth looks at designing the namespace, creating a site topology, designing group policies for locking down client settings, auditing, permissions, backup and recovery, and a look at Microsoft's future direction with Directory Services.

Part III is all about managing Active Directory via automation with Active Directory Service Interfaces (ADSI), ActiveX Data Objects (ADO), and Windows Management Instrumentation (WMI). This section covers how to create and manipulate users, groups, printers, and other objects that you may need in your everyday management of Active Directory. It also describes in depth how you can utilize the strengths of WMI and the .NET System.DirectoryServices namespace to manage Active Directory programmatically via those interfaces.

If you're looking for in-depth coverage of how to use the MMC snap-ins or Resource Kit tools, look elsewhere. However, if you want a book that lays bare the design and management of an enterprise or departmental Active Directory, you need look no further.

Intended Audience

This book is intended for all Active Directory administrators, whether you manage a single server or a global multinational with a farm of thousands of servers. Even if you have the first edition, you'll find a considerable amount of new material in this book, which covers many of the new features in Windows Server 2003. To get the most out of the book, you will probably find it useful to have a server running Windows Server 2003 and the Resource Kit tools available so that you can check out various items as we point them out.

If you have no experience with VBScript, the scripting language we use in Part III, don't worry. The syntax is straightforward, and you should have no difficulty grasping the principles of scripting with ADSI, ADO, and WMI. For those who want to learn more about VBScript, we provide links to various Internet sites and other books as appropriate.

Contents of the Book

This book is split into three parts:

Part I, Active Directory Basics

- Chapter 1 reviews the evolution of the Microsoft NOS and some of the major features and benefits of Active Directory.
- Chapter 2 provides a high-level look at how objects are stored in Active Directory and explains some of the internal structures and concepts that it relies on.
- Chapter 3 reviews the predefined Naming Contexts within Active Directory, what is contained within each, and the purpose of Application Partitions.
- Chapter 4 gives you information on how the blueprint for each object and each object's attributes are stored in Active Directory.
- Chapter 5 details how the actual replication process for data takes place between domain controllers.
- Chapter 6 describes the importance of the Domain Name System (DNS) and what it is used for within Active Directory.
- Chapter 7 gives you a detailed introduction to the capabilities of both user profiles and Group Policy Objects.

Part II, Designing an Active Directory Infrastructure

- Chapter 8 introduces the steps and techniques involved in properly preparing a design that reduces the number of domains and increases administrative control through the use of Organizational Units.
- Chapter 9 shows you how to design a representation of your physical infrastructure within Active Directory to gain very fine-grained control over intrasite and intersite replication.
- Chapter 10 explains how Group Policy Objects function in Active Directory and how you can properly design an Active Directory structure to make the most effective use of these functions.
- Chapter 11 describes how you can design effective security for all areas of your Active Directory, in terms of both access to objects and their properties; it includes information on how to design effective security access logging in any areas you choose.
- Chapter 12 covers procedures for extending the classes and attributes in the Active Directory schema.
-

Conventions in This Book

The following typographical conventions are used in this book:

Constant width

Indicates command-line elements, computer output, and code examples.

Constant width italic

Indicates variables in examples and registry keys.

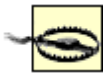
Constant width bold

Indicates user input.

Italic

Introduces new terms and indicates URLs, commands, file extensions, filenames, directory or folder names, and UNC pathnames.

Indicates a tip, suggestion, or general note. For example, we'll tell you if you need to use a particular version or if an operation requires certain privileges.



Indicates a warning or caution. For example, we'll tell you if Active Directory does not behave as you'd expect or if a particular operation has a negative impact on performance.

How to Contact Us

We have tested and verified the information in this book to the best of our ability, but you might find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:
O'Reilly & Associates, Inc. 1005 Gravenstein Highway North Sebastopol, CA 95472 (800) 998-9938 (in the United States or Canada) (707) 829-0515 (international/local) (707) 829-0104 (fax)

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

We have a web page for this book where we list examples and any plans for future editions. You can access this information at:
http://www.oreilly.com/catalog/actdir2

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

Acknowledgments

For the First Edition (Alistair)

Many people have encouraged me in the writing of this book, principally Vicky Launders, my partner, friend, and fountain of useful information, who has been a pinnacle of understanding during all the late nights and early mornings. Without you my life would not be complete.

My parents Pauline and Peter Norris also have encouraged me at every step of the way; many thanks to you both.

For keeping me sane, my thanks go to my good friend Keith Cooper, a natural polymath, superb scientist, and original skeptic; to Steve Joint, for keeping my enthusiasm for Microsoft in check; to Dave and Sue Peace for "Tuesdays," and the ability to look interested in what I was saying and how the book was going no matter how uninterested they must have felt; and to Mike Felmeri for his interest in this book and his eagerness to read an early draft.

I had a lot of help from my colleagues at Leicester University. To Lee Flight, a true networking guru without peer, many thanks for all the discussions, arguments, suggestions, and solutions. I'll remember forever how one morning very early you took the first draft of my 11-chapter book and spread it all over the floor to produce the 21 chapters that now constitute the book. It's so much better for it. Chris Heaton gave many years of dedicated and enjoyable teamwork; you have my thanks. Brian Kerr, who came onto the fast-moving train at high speed, managed to hold on tight through all the twists and turns along the way, and then finally took over the helm. Thanks to Paul Crow for his remarkable work on the Windows 2000 client rollout and GPOs at Leicester. And thanks to Phil Beesley, Carl Nelson, Paul Youngman, and Peter Burnham for all the discussions and arguments along the way. A special thank you goes to Wendy Ferguson for our chats over the past few years.

To the Cormyr crew: Paul Burke, for his in-depth knowledge across all aspects of technology and databases in particular, who really is without peer, and thanks for being so eager to read the book that you were daft enough to take it on your honeymoon; Simon Williams for discussions on enterprise infrastructure consulting and practices, how you can't get the staff these days, and everything else under the sun that came up; Richard Lang for acting as a sounding board for the most complex parts of replication internals, as I struggled to make sense of what was going on; Jason Norton for his constant ability to cheer me up; Mark Newell for his gadgets and Ian Harcombe for his wit, two of the best analyst programmers that I've ever met; and finally, Paul "Vaguely" Buxton for simply being himself. Many thanks to you all.

To Allan Kelly, another analyst programmer par excellence, for various discussions that he probably doesn't remember but that helped in a number of ways.

At Microsoft: Walter Dickson for his insightful ability to get right to the root of any problem, constant accessibility via email and phone, and his desire to make sure that any job is done to the best of its ability; Bob Wells for his personal enthusiasm and interest in what I was doing; Daniel Turner for his help, enthusiasm, and key role in getting Leicester University involved in the Windows 2000 RDP; Oliver Bell for actually getting Leicester University accepted on the Windows 2000 RDP and taking a chance by allocating free consultancy time to the project; Brad Tipp whose enthusiasm and ability galvanized me into action at the U.K. Professional Developers Conference in 1997; Julius Davies for various discussions but among other things telling me how the auditing and permissions aspects of Active Directory had all changed just after I finished the chapter; Karl Noakes, Steve Douglas, Jonathan Phillips, Stuart Hudman, Stuart Okin, Nick McGrath, and Alan Bennett for various discussions.

To Tony Lees, director of Avantek Computer Ltd., for being attentive, thoughtful, and the best all-round salesman I have ever met, many thanks for taking the time to get Leicester University onto the Windows 2000 RDP.

Thanks to Amit D. Chaudhary and Cricket Liu for reviewing parts of the book.

I also would like to thank everyone at O'Reilly but especially my editor Robert Denn for his encouragement, patience, and keen desire to get this book crafted properly.

Part I: Active Directory Basics

This section of the book discusses the basics of Active Directory in order to provide a good grounding in the building blocks and how they function together.

Chapter 1. A Brief Introduction

Active Directory (AD) is Microsoft's network operating system (NOS) directory, built on top of Windows 2000 and Windows Server 2003. It enables administrators to manage enterprise-wide information efficiently from a central repository that can be globally distributed. Once information about users and groups, computers and printers, and applications and services has been added to Active Directory, it can be made available for use throughout the entire network to as many or as few people as you like. The structure of the information can match the structure of your organization, and your users can query Active Directory to find the location of a printer or the email address of a colleague. With Organizational Units, you can delegate control and management of the data however you see fit. If you are like most organizations, you may have a significant amount of data (e.g., thousands of employees or computers). This may seem daunting to enter in Active Directory, but fortunately Microsoft has some very robust yet easy-to-use Application Programming Interfaces (APIs) to help facilitate data management programmatically.

This book is an introduction to Active Directory, but an introduction with a broad scope. In Part I, we cover many of the basic concepts within Active Directory to give you a good grounding in some of the fundamentals that every administrator should understand. In Part II, we focus on various design issues and methodologies, to enable you to map your organization's business requirements into your Active Directory infrastructure. Getting the design right the first time around is critical to a successful implementation, but it can be extremely difficult if you have no experience deploying Active Directory. In Part III, we cover in detail management of Active Directory programmatically through scripts based on Active Directory Service Interfaces (ADSI), ActiveX Data Objects (ADO), and Windows Management Instrumentation (WMI). No matter how good your design is, unless you can automate your environment, problems will creep in, causing decreased uniformity and reliability.

Before moving on to some of the basic components within Active Directory, we will now review how Microsoft came to the point of implementing an LDAP-based directory service to support their NOS environment.

1.1 Evolution of the Microsoft NOS

"NOS" is the term used to describe a networked environment in which various types of resources, such as user, group, and computer accounts, are stored in a central repository that is controlled and accessible to end users. Typically a NOS environment is comprised of one or more servers that provide NOS services, such as authentication and account manipulation, and multiple end users that access those services.

Microsoft's first integrated NOS environment became available in 1990 with the release of Windows NT 3.0, which combined many features of the LAN Manager protocols and OS/2 operating system. The NT NOS slowly evolved over the next eight years until Active Directory was first released in beta in 1997.

Under Windows NT, the "domain" concept was introduced, providing a way to group resources based on administrative and security boundaries. NT domains are flat structures limited to about 40,000 objects (users, groups, and computers). For large organizations, this limitation imposed superficial boundaries on the design of the domain structure. Often, domains were geographically limited as well because the replication of data between domain controllers (i.e., servers providing the NOS services to end users) performed poorly over high-latency or low-bandwidth links. Another significant problem with the NT NOS was delegation of administration, which typically tended to be an all-or-nothing matter at the domain level.

Microsoft was well aware of these limitations and needed to rearchitect their NOS model into something that would be much more scalable and flexible. For that reason, they looked to LDAP-based directory services as a possible solution.

1.1.1 Brief History of Directories

In generic terms, a directory service is a repository of network, application, or NOS information that is useful to multiple applications or users. Under this definition, the Windows NT NOS is a type of directory service. In fact, there are many different types of directories, including Internet white pages, email systems, and even the Domain Name System (DNS). While each of these systems have characteristics of a directory service, X.500 and the Lightweight Directory Access Protocol (LDAP) define the standards for how a true directory service is implemented and accessed.

In 1988, the International Telecommunication Union (ITU) and International Organization of Standardization (ISO) teamed up to develop a series of standards around directory services, which has come to be known as X.500. While X.500 proved to be a good model for structuring a directory and provided a lot of functionality around advanced operations and security, it was difficult to implement clients to utilize it. One reason is that X.500 is based on the OSI (Open System Interconnection) protocol stack instead of TCP/IP, which had become the standard for the Internet. The X.500 directory access protocol (DAP) was very complex and implemented a lot of features most clients never needed. This prevented large-scale adoption. It is for this reason that a group headed by the University of Michigan started work on a "lightweight" X.500 access protocol that would make X.500 easier to utilize.

The first version of the Lightweight Directory Access Protocol (LDAP) was released in 1993 as RFC 1487, but due to the absence of many features provided by X.500, it never really took off. It wasn't until LDAPv2 was released in 1995 as RFC 1777 that LDAP started to gain popularity. Prior to LDAPv2, the primary use of LDAP was as a gateway between X.500 servers. Simplified clients would interface with the LDAP gateway, which would translate the requests and submit it to the X.500 server. The University of Michigan team thought that if LDAP could provide most of the functionality necessary to most clients, they could remove the middleman (the gateway) and develop an LDAP-enabled directory server. This directory server could use many of the concepts from X.500, including the data model, but would leave out all the overheard provided by the numerous features it implemented. Thus the first LDAP directory server was released in late 1995 by the University of Michigan team, and it turned into the basis for many future directory servers.

In 1997, the last major update to the LDAP specification was described in RFC 2251. It provided several new features and made LDAP robust enough and extensible enough to be suitable for most vendors to implement. Since then, companies such as Netscape, Sun, Novell, and Microsoft have developed LDAP-based directory servers. Most recently, RFC 3377 was released, which summarizes all of the major LDAP RFCs.

1.2 Windows NT Versus Active Directory

As we mentioned earlier, Windows NT and Active Directory both provide directory services to clients (Windows NT in a more generic sense). And while both share some common concepts, such as Security Identifiers (SIDs) to identify security principals, they are very different from a feature, scalability, and functionality point of view. Table 1-1 contains a comparison of features between Windows NT and Active Directory.

Table 1-1. A comparison between Windows NT and Active Directory

Windows NT	**Active Directory**
Single-master replication is used, from the PDC master to the BDC subordinates.	Multimaster replication is used between all domain controllers.
Domain is the smallest unit of partitioning.	Naming Contexts and Application Partitions are the smallest unit of partitioning.
System policies can be used locally on machines or set at the domain level.	Group policies can be managed centrally and used by clients throughout the forest based on domain, site or OU criteria.
Data cannot be stored hierarchically within a domain.	Data can be stored in a hierarchical manner using OUs.
Domain is the smallest unit of security delegation and administration.	A property of an object is the smallest unit of security delegation/administration.
NetBIOS and WINS used for name resolution.	DNS is used for name resolution.
Object is the smallest unit of replication.	Attribute is the smallest unit of replication. In Windows Server 2003 Active Directory, some attributes replicate on a per-value basis (such as the member attribute of group objects).
Maximum recommended database size for SAM is 40 MB.	Recommended maximum database size for Active Directory is 70 TB.
Maximum effective number of users is 40,000 (if you accept the recommended 40 MB maximum).	The maximum number of objects is in the tens of millions.
Four domain models (single, single-master, multimaster, complete-trust) required to solve per-domain admin-boundary and user-limit problems.	No domain models required as the complete-trust model is implemented. One-way trusts can be implemented manually.
Schema is not extensible.	Schema is fully extensible.
Data can only be accessed through a Microsoft API.	Supports LDAP, which is the standard protocol used by directories, applications, and clients that want to access directory data. Allows for cross-platform data access

1.3 Windows 2000 Versus Windows Server 2003

While the first version of Active Directory available with Windows 2000 was very stable and feature-rich, it still had room for improvement, primarily around usability and performance. With Windows Server 2003, Microsoft has addressed many of these issues. To utilize these features you have to upgrade your domain controllers to Windows Server 2003 and raise the domain and forest functional levels as necessary.

The difference between Windows 2000 Active Directory and Windows Server 2003 Active Directory is more evolutionary than revolutionary. The decision to upgrade to Windows Server 2003 is a subjective one, based on your needs. For example, if you have a lot of domain controllers and Active Directory sites, you may want to take advantage of the improvements with replication as soon as possible. Or perhaps you've been dying to rename a domain, a capability available in Windows Server 2003 Active Directory. On the whole, Microsoft added or updated more than 100 features within Active Directory, and we will now discuss some of the more significant ones.

For more information on migrating to Windows Server 2003 from Windows 2000 check out Chapter 14.

Some of the new features are available as soon as you promote the first Windows Server 2003 domain controller into an existing Windows 2000 Active Directory domain. In Table 1-2, the features available when you do so are listed along with descriptions. Note that these features will apply only to the Windows Server 2003 domain controllers in the domain.

Table 1-2. Windows 2000 domain functional level feature list

Feature	Description
Application Partitions	You can create your own partitions to store data separately from the default partitions, and you can configure which DCs in the forest replicate it.
GC not required for logon (i.e., universal group caching)	Under Windows 2000, a DC had to contact a GC to determine universal group membership and subsequently to allow users to logon. This feature allows DCs to cache universal group membership so that it is not necessary to contact a GC for logins.
MMC enhancements and new command-line tools	The new Active Directory Users and Computers allows you to save queries, drag and drop, and edit multiple users at once, and it is much more efficient about scrolling through a large number of objects. In addition, several new command-line tools (dsadd, dsmod, dsrm, dsquery, dsget, and dsmove) come installed with the server, allowing for greater flexibility in managing Active Directory.
Install from media	Administrators can create new DCs for an existing domain by installing from a backup of an existing DC that resides on media such as a CD or DVD.
	You can apply a WMI filter, which is a query that can utilize any WMI information on a client, to a GPO, and

1.4 Summary

This chapter has been a brief introduction to the origins of Active Directory and some of the new features available in Windows Server 2003. The rest of the chapters in Part I will cover the conceptual introduction to Active Directory and equip you to get the most out of Part II and Part III.

Chapter 2. Active Directory Fundamentals

This chapter aims to bring you up to speed on the basic concepts and terminology used with Active Directory. It is important to understand each component of Active Directory before embarking on a design, or your design may leave out a critical element.

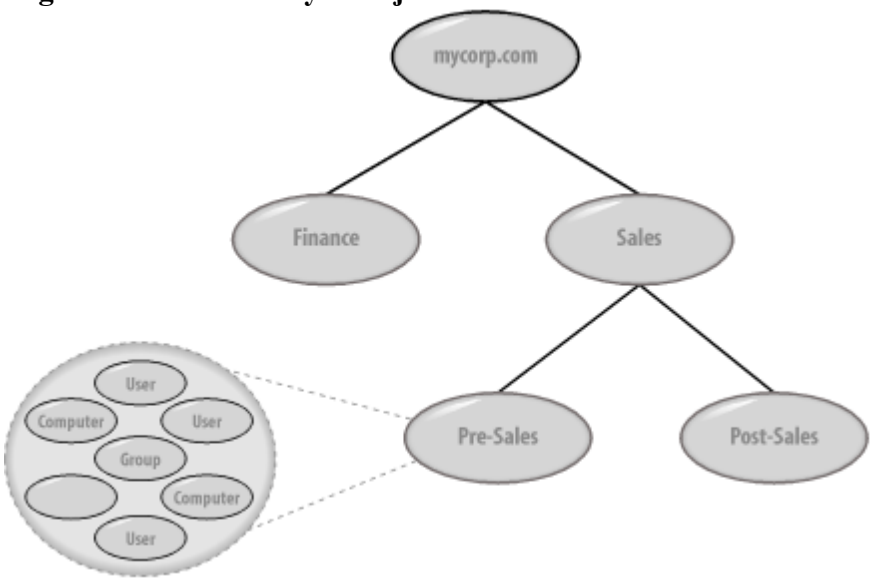
2.1 How Objects Are Stored and Identified

Data is stored within Active Directory in a hierarchical fashion similar to the way data is stored in a filesystem. Each entry is referred to as an object. At the structural level, there are two types of objects: containers and non-containers, also known as leaf nodes. One or more containers branch off in a hierarchical fashion from a root container. Each container may contain leaf nodes or other containers. A leaf node, however, as the name implies, may not contain any other objects.

Consider the parent-child relationships of the containers and leaves in Figure 2-1. The root of this tree has two children, Finance and Sales. Both of these are containers of other objects. Sales has two children of its own, Pre-Sales and Post-Sales. Only the Pre-Sales container is shown as containing additional child objects. The Pre-Sales container holds user, group, and computer objects as an example.[1] Each of these child nodes is said to have the Pre-Sales container as its parent. Figure 2-1 represents what is known in Active Directory as a domain.

[1] User, group, and computer objects are actually containers, as they can contain other objects such as printers. However, they are not normally drawn as containers in domain diagrams such as this.

Figure 2-1. A hierarchy of objects



The most common type of container you will create in Active Directory is an Organizational Unit, but there are others as well, such as the one called Container. Each of these has its place, as we'll show later, but the one that we will be using most frequently is the Organizational Unit (OU).

2.1.1 Uniquely Identifying Objects

When you are potentially storing millions of objects in Active Directory, each object has to be uniquely locatable and identifiable. To that end, objects have a Globally Unique Identifier (GUID) assigned to them by the system at creation. This 128-bit number is guaranteed to be unique by Microsoft. The object GUID stays with the object until it is deleted, regardless of whether it is renamed or moved within the Directory Information Tree (DIT).

While an object GUID is unique and resilient, it is not very easy to remember, nor is it based on the directory hierarchy. For that reason, another way to reference objects, called an ADsPath, is more commonly used.

2.1.1.1 ADsPaths

Hierarchical paths in Active Directory are known as ADsPaths and can be used to uniquely reference an object. In fact, ADsPath is a slightly more general term and is used by Microsoft to apply to any path to any of the major directories: Active Directory, Windows NT, Novell's NDS, and many others.

ADsPaths for Active Directory objects are normally represented using the syntax and rules defined in the LDAP standards. Let's take a look at how a path to the root of Figure 2-1 looks:

2.2 Building Blocks

Now that we've shown how objects are structured and referenced, let's look at the core concepts behind Active Directory.

2.2.1 Domains and Domain Trees

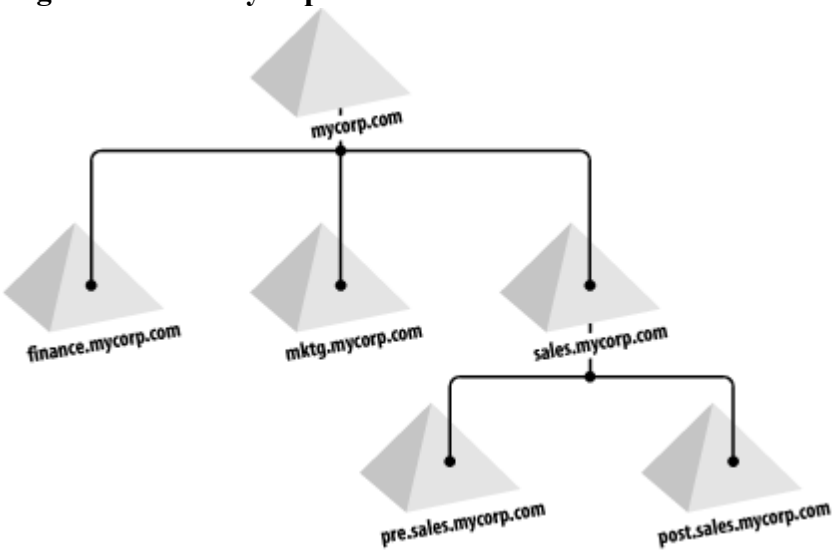
Active Directory's logical structure is built around the concept of domains introduced in Windows NT 3.x and 4.0. However, in Active Directory, domains have been updated significantly from the flat and inflexible structure imposed by Windows NT. An Active Directory domain is made up of the following components:

- An X.500-based hierarchical structure of containers and objects
- A DNS domain name as a unique identifier
- A security service, which authenticates any access to resources via accounts in the domain or trusts with other domains
- One or more policies that dictate how functionality is restricted for users or machines within that domain

A domain controller (DC) can be authoritative for one and only one domain. Currently it is not possible to host multiple domains on a single DC. For example, Mycorp Company has already been allocated a DNS domain name for their company called mycorp.com, so they decide that the first Active Directory domain that they are going to build is to be named mycorp.com. However, this is only the first domain in a series that needs to be created, and mycorp.com is in fact the root of a domain tree.

The mycorp.com domain itself, ignoring its contents, is automatically created as the root node of a hierarchical structure called a domain tree. This is literally a series of domains connected together in a hierarchical fashion, all using a contiguous naming scheme. So, when Finance, Marketing, and Sales each wants its own domain, the names become finance.mycorp.com, mktg.mycorp.com, and sales.mycorp.com. Each domain tree is called by the name given to the root of the tree; hence, this domain tree is known as the mycorp.com tree, as illustrated in Figure 2-2. You can also see that we have added further domains below sales, for pre-sales and post-sales.

Figure 2-2. The mycorp.com domain tree



You can see that in Mycorp's setup, we now have a contiguous set of domains that all fit into a neat tree. Even if we had only one domain, it would still be a domain tree, albeit with only one domain.

2.3 Summary

In this chapter, we've gone over the groundwork for some of the main internals of Active Directory. We covered such concepts as domains, trees, forests, Organizational Units, the Global Catalog, FSMOs, Windows 2000 domain modes, and Windows Server 2003 functional levels. We then delved into how groups work in Active Directory and what features are available under the various domain modes and functional levels.

With this information under our belts, let's now take a look at how data is organized in Active Directory with Naming Contexts and Application Partitions.

Chapter 3. Naming Contexts and Application Partitions

Due to the distributed nature of Active Directory, it is necessary to segregate data into partitions. If data partitioning were not used, every domain controller would have to replicate all the data within a forest. Often it is advantageous to group data based on geographical or political requirements. Think of a domain as a big data partition, which is also referred to as a naming context (NC). Only domain controllers that are authoritative for a domain need to replicate the information within it. On the other hand, there is some Active Directory data that must be replicated to all domain controllers. There are three predefined naming contexts within Active Directory:

- A Domain Naming Context for each domain
- The Configuration Naming Context for the forest
- The Schema Naming Context for the forest

Each of these naming contexts represents a different aspect of Active Directory data. The Configuration NC holds data pertaining to the configuration of the forest, for example, the objects representing naming contexts, LDAP policies, sites, subnets, and so on. The Schema NC contains the set of object class and attribute definitions for the types of data that can be stored in Active Directory. Each domain in a forest also has a Domain NC, which contains data specific to the domain, for example, users, groups, computers, etc.

In Windows Server 2003 Active Directory, Microsoft extended the naming context concept by allowing user-defined partitions called application partitions. Application partitions can contain any type of object except security principals, such as user objects. The major benefit of application partitions is that administrators can define which domain controllers replicate the data contained within them. Application partitions are not restricted by domain boundaries, as is the case with Domain NCs.

You can retrieve a list of the naming contexts and application partitions a specific domain controller maintains by querying its Root DSE entry. You can view the Root DSE by opening the LDP utility, which is available from the Windows Support Tools. Select Connection ⟶ Connect from the menu, enter the name of a domain controller, and click OK. The following attributes pertain to naming contexts and application partitions:

namingContexts

List of DNs of all the naming contexts and application partitions maintained by the DC.

defaultNamingContext

DN of the Domain NC the DC is authoritative for.

configurationNamingContext

DN of the Configuration NC.

schemaNamingContext

DN of the Schema NC.

rootNamingContext

DN of the Domain NC for the forest root domain.

In this chapter, we will review each of the three predefined naming contexts and describe the data contained within each, and then cover application partitions and example uses.

3.1 Domain Naming Context

Each Active Directory domain is represented by a Domain NC, which holds the domain-specific data. The root of this NC is represented by a domain's distinguished name (DN). For example, the mycorp.com domain's DN would be dc=mycorp,dc=com. Each domain controller in the domain replicates a copy of the Domain NC.

Table 3-1 contains a list of the default top-level containers found in a Domain NC. Note that to see all of these containers with the Active Directory Users and Computers (ADUC) snap-in, you must select View ⟶ Advanced Features from the menu. Alternatively, you can browse all of these containers with the ADSI Edit tool available in the Windows Support Tools on any Windows Server 2003 or Windows 2000 CD.

Table 3-1. Default top-level containers of a Domain NC

Relative distinguished name	Description
cn=Builtin	Container for predefined built-in local security groups. Examples include Administrators, Users and Account Operators.
cn=Computers	Default container for computer objects representing member servers and workstations.
ou=Domain Controllers	Default organizational unit for computer objects representing domain controllers.
cn=ForeignSecurityPrincipals	Container for placeholder objects representing members of groups in the domain that are from a domain external to the forest.
cn=LostandFound	Container for orphaned objects.
cn=NTDS Quotas	Container to store quota objects, which are used to restrict the number of objects a security principal can create in a partition or container. This container is new in Windows Server 2003.
cn=Program Data	Container for applications to store data instead of using a custom top-level container. This container is new in Windows Server 2003.
cn=System	Container for miscellaneous domain configuration objects. Examples include trust objects, DNS objects, and group policy objects.
cn=Users	Default container for user and group objects.

3.2 Configuration Naming Context

The Configuration NC is the primary repository for configuration information for a forest. Every domain controller in the forest replicates the Configuration NC, which is why it is considered forest-wide. The root of the Configuration NC is found in the Configuration container, which is a subcontainer of the forest root domain. For example, the mycorp.com forest would have a Configuration NC located at cn=configuration,dc=mycorp,dc=com.

Table 3-2 contains a list of the default top-level containers found in the Configuration NC.

Table 3-2. Default top-level containers of the Configuration NC

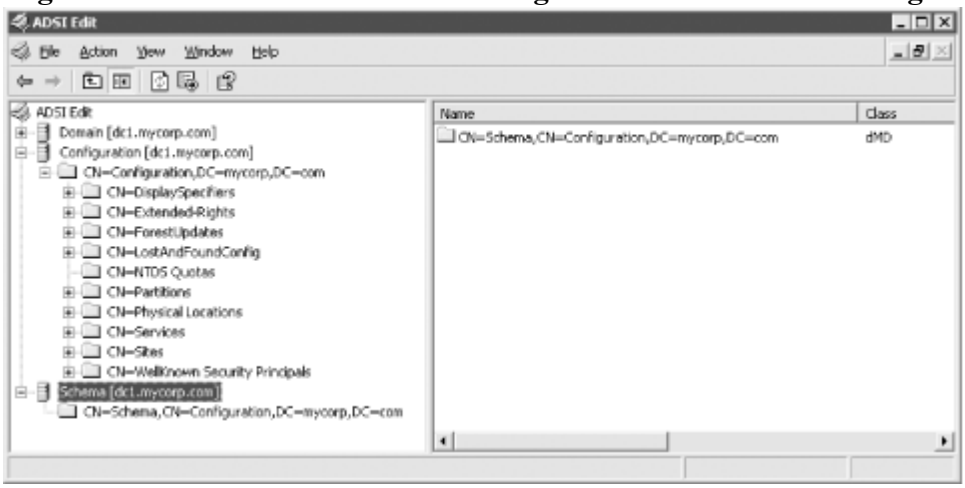
Relative Distinguished Name	**Description**
cn=DisplaySpecifiers	Container that holds display specifier objects, which define various properties and functions of the Active Directory MMC Snap-ins.
cn=Extended-Rights	Container for extended rights (controlAccessRight) objects.
cn=ForestUpdates	Contains objects that are used to represent the state of forest and domain functional level changes. This container is new in Windows Server 2003.
cn=LostandFoundConfig	Container for orphaned objects.
cn=NTDS Quotas	Container to store quota objects, which are used to restrict the number of objects that security principals can create in a partition or container. This container is new in Windows Server 2003.
cn=Partitions	Contains objects for each naming context, application partition, and external reference.
cn=Physical Locations	Contains location objects (physicalLocation), which can be associated with other objects to denote location of the object.
cn=Services	Store of configuration information about services such as FRS, Exchange, and even Active Directory itself.
cn=Sites	Contains all of the site topology and replication objects. This includes site, subnet, siteLink, server and nTDSConnection objects, to name a few.
cn=WellKnown Security Principals	Holds objects representing commonly used foreign security principals, such as Everyone, Interactive, and Authenticated Users.

3.3 Schema Naming Context

The Schema NC contains objects representing the classes and attributes that Active Directory supports. The schema is defined on a forest-wide basis, so the Schema NC is replicated to every domain controller in the forest. The root of the Schema NC can be found in the Schema container, which is a subcontainer of the Configuration container. For example, in the mycorp.com forest, the Schema NC would be located at cn=schema,cn=configuration,dc=mycorp,dc=com.

Although the Schema container appears to be a child of the Configuration container, it is actually a separate naming context in its own right. Figure 3-1 shows how the Schema and Configuration NCs are segregated in the ADSI Edit tool.

Figure 3-1. ADSI Edit view of the Configuration and Schema Naming Contexts



You may be wondering why the schema isn't just contained within the Configuration NC. As we covered in Chapter 2, there is a Schema FSMO role that is the single master for updates to schema objects. The Schema FSMO role is necessary due to the highly sensitive nature of the schema and the fact that two conflicting schema updates could spell disaster for a forest. Since there is only a single domain controller that schema changes can be made on, the schema must replicate differently from the Configuration NC, which can be updated by any domain controller in the forest.

Unlike the Domain and Configuration NCs, the Schema NC does not contain a hierarchy of containers or organizational units. Instead it is a single container that has classSchema, attributeSchema, and subSchema objects. The classSchema objects define the different types of classes and their associated attributes. The attributeSchema objects define all the attributes that are used as part of classSchema definitions. There is also a single subSchema object that represents the abstract schema as defined in the LDAPv3 RFC (http://www.ietf.org/rfc/rfc2254.txt).

Chapter 4 and Chapter 12 deal with the schema in more depth.

3.4 Application Partitions

Application partitions are a new feature in Windows Server 2003. They enable administrators to create areas in Active Directory to store data on DCs they choose rather than on every DC in a domain or forest. You can define which domain controllers hold a copy of the partition, known as a replica. There is no limitation based on domain or site membership, which means you can configure any domain controller in a forest to hold any application partition replica. The existing site topology will be used to automatically create the necessary connection objects to replicate among the servers that hold replicas of an application partition. Domain controllers will also register the necessary SRV records (explained in more detail in Chapter 6), so that clients can use the DC locator process to find the optimal domain controller for an application partition, just as they would for a domain.

There are a few limitations to be aware of with application partitions:

- Application partitions cannot contain security principals, which most notably includes user, group, and computer objects. Any other type of object can be created in an application partition.
- None of the objects contained in an application partition are replicated to the global catalog. Even if a domain controller that holds a replica of an application partition is also a global catalog server, the domain controller will not return any objects from the application partition during a global catalog search.
- Objects in an application partition cannot be moved outside the partition. This is different than objects contained in domains, which can be moved between domains.
- The Domain Naming FSMO must be on a Windows Server 2003 domain controller to create an application partition. After the application partition has been created, you can move the Domain Naming FSMO back to a Windows 2000 domain controller if necessary.

Application partitions are named similarly to domains. For example, if you created an application partition called "apps" directly under the mycorp.com domain, the DNS name would be apps.mycorp.com and the distinguished name would be dc=apps,dc=mycorp,dc=com. Application partitions can be rooted under domains, as shown in the previous example, nested under other application partitions (for example, dc=sales,dc=apps,dc=mycorp,dc=com) or as part of a new domain tree (for example, dc=apps,dc=local). For more information on creating and managing application partitions, check out the NTDSUTIL utility.

Application partitions tend to store dynamic data—data with a limited lifespan. See the next section for more on this. Dynamic data from network services such as DNS, Dynamic Host Configuration Protocol (DHCP), Common Open Policy Service (COPS), Remote Access Service (RAS), and RADIUS can all reside in a partition in AD. This allows uniformity of access from applications via a single methodology. This enables developers to write to a special area only available to specific servers rather than into a domain partition that is replicated to every DC. In fact, application partitions will allow multiple versions of COM+ applications to be installed and configured on the same computer, resulting in more cost-effective management of server applications.

3.4.1 Storing Dynamic Data

While application partitions give administrators more control over how to replicate application data, the problem of data cleanup still exists. That is, applications that add data to Active Directory are not always good about cleaning it up after it is no longer needed. That's why the ability to create dynamic data was also added as a feature in Windows Server 2003 Active Directory. Dynamic objects are objects that have a time-to-live (TTL) value that determines how long the object will exist before being automatically deleted by Active Directory. Dynamic objects typically have a fairly short life span (i.e., days). An example use of dynamic objects is an e-commerce website that needs to store user session information temporarily. Since a directory is likely going to be where the user profile information resides, it can be advantageous to use the same store for session-based information, which is generally short-lived. The default

3.5 Summary

In this chapter, we covered how objects are grouped at a high level into naming contexts and application partitions, which are used as replication boundaries. The Domain NC contains domain-specific data such as users, groups, and computers. The Configuration NC contains forest-wide configuration data such as the site topology objects and objects that represent naming contexts and application partitions. The Schema NC contains all the schema objects that define how data is structured and represented in Active Directory. Application partitions were introduced in Windows Server 2003 Active Directory as a way for administrators to define their own grouping of objects and, subsequently, replication boundaries. Storage of DNS data for AD-Integrated DNS zones is the classic example of when it makes sense to use application partitions, due to the increased control they give you over which domain controllers replicate the data. Dynamic objects are also new to Windows Server 2003 Active Directory; they allow you to create objects that have a time-to-live (TTL) value. After the TTL expires, Active Directory automatically deletes the object.

Chapter 4. Active Directory Schema

The schema is the blueprint for data storage in Active Directory. Each object in Active Directory is an instance of a class in the schema. A user object, for example, exists as an instance of the user class. Attributes define the pieces of information that a class, and thus an instance of that class, can hold. Syntaxes define the type of data that can be placed into an attribute. As an example, if an attribute is defined with a syntax of Boolean, it can store True or False as its value.

Active Directory contains many attributes and classes in the default schema, some of which are based on standards and some of which Microsoft needed for its own use. However, the Active Directory schema was designed to be extensible, so that administrators could add any classes or attributes they deem necessary. In fact, extending the schema is not a difficult task; it is often more difficult to design the changes that you would like to incorporate. Schema design issues are covered in Chapter 12, and in Chapter 24 we cover how to extend the schema programmatically. In this chapter, we're concerned only with the fundamentals of the schema.

4.1 Structure of the Schema

The Schema Container is located in Active Directory under the Configuration Container. For example, the distinguished name of the Schema Container in the mycorp.com forest would be cn=schema,cn=Configuration,dc=mycorp,dc=com. You can view the contents of the container directly by pointing an Active Directory viewer such as ADSI Edit or LDP at it. You can also use the Active Directory Schema MMC snap-in, which splits the classes and attributes in separate containers for easy viewing, even though in reality all the schema objects are stored directly in the Schema Container.

The schema itself is made up of two types of Active Directory objects: classes and attributes. In Active Directory, these are known respectively as classSchema (Class-Schema) and attributeSchema (Attribute-Schema) objects. The two distinct forms of the same names result from the fact that the cn (Common-Name) attribute of a class contains the hyphenated easy-to-read name of the class, and the lDAPDisplayName (LDAP-Display-Name) attribute of a class contains the concatenated string format that is used when querying Active Directory with LDAP or ADSI. In the schema, the lDAPDisplayName attribute of each object is normally made by capitalizing the first letter of each word of the Common-Name, then removing the hyphens and concatenating all the words together. Finally, the first letter is made lowercase.[1] This creates simple names like user, as well as the more unusual sAMAccountName and lDAPDisplayName. We'll specify the more commonly used LDAP display name format from now on.

[1] Names defined by the X.500 standard don't tend to follow this method. For example, the Common-Name attribute has an LDAP-Display-Name of cn, and the Surname attribute has an LDAP-Display-Name of sn.

Whenever you need to create new types of objects in Active Directory, you must first create a classSchema object defining the class of the object and the attributes it contains. Once the class is properly designed and added to the schema, you can then create objects in Active Directory that use the class. Alternatively, if you want to add a new attribute to an object, you must first create the attributeSchema object and associate the attribute with whatever classes you want to use it with.

Before we delve into what makes up an Active Directory class or attribute, we need to explain how each class that you create is unique not just within your Active Directory but also throughout the world.

4.1.1 X.500 and the OID Namespace

Active Directory is based on LDAP, which was originally based on the X.500 standard created by the ISO (International Organization for Standardization) and ITU (International Telecommunications Union) organizations in 1988. To properly understand how the Active Directory schema works, you really need to understand the basics of X.500; we'll run through them next.

The X.500 standard specifies that individual object classes in an organization can be uniquely defined using a special identifying process. The process has to be able to take into account the fact that classes can inherit from one another, as well as the potential need for any organization in the world to define and export a class of their own design.

To that end, the X.500 standard defined an Object Identifier (OID) to uniquely identify every schema object. This OID is composed of two parts:

- One to indicate the unique path to the branch holding the object in the X.500 treelike structure
- Another to indicate the object uniquely in that branch

OID notation uses integers for each branch and object, as in the following example OID for an object:

```
1.3.6.1.4.1.3385.12.497
```

This uniquely references object 497 in branch 1.3.6.1.4.1.3385.12. The 1.3.6.1.4.1.3385.12 branch is contained in a branch whose OID is 1.3.6.1.4.1.3385, and so on.

4.2 Attributes (attributeSchema Objects)

Just as class information is stored in Active Directory as instances of the class called classSchema, attributes are represented by instances of the class called attributeSchema. As with all objects, the attributeSchema class has a number of attributes that can be set when specifying a new instance. The attributeSchema class inherits attributes from the class called Top. However, most of the Top attributes are not really relevant here. Table 4-1 shows the defining attributes of an instance of the attributeSchema class (i.e., an attribute) that can be set.

Table 4-1. The defining attributes of an attributeSchema object instance

Attribute	Syntax	Mandatory	Multivalued	Description
attributeId	OID	Yes	No	The OID that uniquely identifies this attribute.
cn	Unicode string	Yes	No	The Relative Distinguished Name (RDN).
isSingleValued	Boolean	Yes	No	Whether this attribute is multivalued.
lDAPDisplayName	Unicode string	Yes	No	The name by which LDAP clients identify this attribute.
attributeSyntax	OID	Yes	No	Half of a pair of properties that define the syntax of an attribute. This one is an OID.
oMSyntax	Integer	Yes	No	Half of a pair of properties that define the syntax of an attribute. This one is an integer.
schemaIDGUID	Octet string	Yes	No	Globally Unique Identifier (GUID) to uniquely identify this attribute.
objectClass	OID	Yes	Yes	This will hold the values "attributeSchema" and "Top" to indicate that the value is an instance of those classes.

4.3 Attribute Syntax

The syntax of an attribute represents the kind of data it can hold; people with a programming background are probably more familiar with the term "data type." Unlike attributes and classes, the supported syntaxes are not represented as objects in Active Directory. Instead, Microsoft has coded these syntaxes internally into Active Directory itself. Consequently, any new attributes you create in the schema must use one of the predefined syntaxes.

Whenever you create a new attribute, you must specify its syntax. To uniquely identify the syntax among the total set of 21 syntaxes, you must specify 2 pieces of information: the OID of the syntax and a so-called OM syntax. This pair of values must be set together and correctly correlate with Table 4-3. More than one syntax has the same OID, which may seem strange; and to distinguish between different syntaxes uniquely, you thus need a second identifier. This is the result of Microsoft requiring some syntaxes that X.500 did not provide. Table 4-3 shows the 21 expanded syntaxes, including the name of the syntax with alternate names followed in parentheses.

Table 4-3. Syntax definitions

Syntax	OID	OM syntax	Description
Undefined	2.5.5.0	N/A	Not a valid syntax
Distinguished Name	2.5.5.1	127	The Fully Qualified Domain Name (FQDN) of an object in Active Directory
Object ID	2.5.5.2	6	OID
Case-sensitive string	2.5.5.3	20	A string that differentiates between uppercase and lowercase
Case-insensitive string	2.5.5.4	20	A string that does not differentiate between uppercase and lowercase
Print case string (Printable-String)	2.5.5.5	19	A normal printable string
Print case string (IA5-String)	2.5.5.5	22	A normal printable string
Numeric string	2.5.5.6	18	A string of digits
OR name	2.5.5.7	127	An X.400 email address
Boolean	2.5.5.8	1	True or false
Integer (integer)	2.5.5.9	2	A 32-bit number
Integer (enumeration)	2.5.5.9	10	A 32-bit number

4.4 Classes (classSchema Objects)

Schema classes are defined as instances of the classSchema class. Table 4-4 shows the most important attributes that you may wish to set.

Table 4-4. The defining attributes of a classSchema object instance

Attribute	Syntax	Mandatory	Multi-valued	Description
cn	Unicode	Yes	No	The Relative Distinguished Name (RDN).
governsID	OID	Yes	No	The OID that uniquely identifies objects of this class.
lDAPDisplayName	Unicode	No	No	The name by which LDAP clients identify this class.
schemaIDGUID	Octet string	Yes	No	Globally Unique Identifier (GUID) to uniquely identify this class.
rDNAttID	OID	No	No	The attribute that indicates what two-letter-prefix (cn=, ou=, dc=) is used to reference the class. You should use only cn here unless you have a very solid idea of what you are doing and why.
description	Unicode string	No	No	A description of the attribute.
subClassOf	OID	Yes	No	The class that this one inherits from; the default is Top.[3]
mustContain	OID	No	Yes	The list of attributes that are mandatory for this class.
systemMustContain	OID	No	Yes	System version of the previous attribute.

4.5 Summary

In this chapter we've shown you how the internal blueprint for all objects in Active Directory, known as the schema, was derived from the X.500 directory service. We explained the purpose of the OID numbering system and how it can be used. We then detailed how an attribute and its syntax is structured in the schema as attributeSchema objects, using the userPrincipalName attribute as an example. We showed how attributes are added to classes by detailing how classes are stored in the schema as instances of classSchema objects. To make this clearer, we dug into the details of the user class to see how it was constructed. Finally, we covered how auxiliary classes can be dynamically linked in Windows Server 2003 and why it is significant.

Chapter 12 builds on what you've learned here to demonstrate how you can design and implement schema extensions.

Chapter 5. Site Topology and Replication

This chapter introduces a major feature of Active Directory: multi-master replication. Active Directory was one of the first LDAP-based directories to offer multi-master replication. Most directories replicate data from a single master server to subordinate servers. This is how replication worked in Windows NT 4.0 as an example. Obviously, there are several problems with a single-master replication scheme, including single point of failure for updates, geographic distance from master to clients performing the updates, and less efficient replication due to single originating location of updates. Active Directory replication addresses these issues, but with a price. To get the benefit of a multi-master replication, you must first create a site topology that defines how domain controllers should replicate with each other. Especially in large environments, maintaining a site topology can be a significant amount of overhead.

This chapter looks at the basics of how sites and replication work in Active Directory. In Chapter 9, we'll describe the physical infrastructure of a network layout using sites. We'll also discuss in that chapter how the Knowledge Consistency Checker (KCC) sets up and manages the replication connections and details on how to effectively design and tailor sites, site links, and replication in Active Directory.

5.1 Site Topology

Active Directory uses the term site to mean a collection of subnets that coexist on a local area network (LAN) or metropolitan area network (MAN), i.e., a physical network in a particular location with good connectivity between all sections of that network. Active Directory uses sites to define boundaries of replication around the physical network.

Active Directory replication is very efficient. Only changed attributes are replicated, rather than entire objects, as was the case in Windows NT. And with Windows Server 2003, link-value replication is available for some attributes, so only changed values for a multi-valued attribute are replicated instead of all values. Link-value replication is a much needed feature which was not available in Windows 2000 Active Directory; it is intended to address issues such as the 5,000 member limitation for group objects. Replication also can take place over multiple TCP/IP transports, so that you can find a replication protocol to suit the environment a particular site requires.

The recommended minimum speed for a well-connected network is 1.5 Mbps (i.e., a T1 link). You will see this actual value vary from article to article and book to book, as different people find that their network runs fine over a slower connection speed. We'll cover this later, but the absolute true minimum is around 128 Kbps of available replication bandwidth out of a 256 Kbps total available bandwidth. Your mileage may vary; the only way to determine the best solution in your environment is by testing.

Administrators must create the site topology in Active Directory, as the process is not automatic. The main site-topology objects of interest include the site objects, subnet objects, and site link objects. One of the major uses of the site topology is for clients to find their closest DC. That is why subnet information must be associated with sites. Clients use their IP address to determine which Active Directory subnet they belong to and subsequently which site. The site information can then be used to determine the closest DC.

Once you've set up a site, an Active Directory process called the Knowledge Consistency Checker (KCC) automatically creates and dynamically manages a replication schedule and a set of intrasite (i.e., within a site) replication links among DCs in the site. As you add more DCs, more intrasite links are added automatically. If you were to do nothing more, data would be effectively replicated by Active Directory around your site. When you add your second site, the same automatic intrasite creation mechanisms spring into action, creating links and a replication schedule among the various DCs in this second site. The algorithm that is used adapts as more sites and DCs are added, so that certain built-in criteria are never breached; this assures that the network is always properly replicated. Note, however, that creating a second site does not trigger the system to also automatically create intersite (i.e., between sites) replication links and a replication schedule. Instead, site links that connect two sites have to be created manually. We'll cover the KCC in greater depth later in Chapter 9.

5.1.1 Site and Replication Management Tools

Obviously, as more sites and connections are created, the topology can get very large. Microsoft provides the Active Directory Sites and Services snap-in to help manage the topology. It actually allows you to get right into the guts of the Sites Container, which holds all the site topology objects and connection objects. The Sites Container is located directly under the Configuration Container in the Configuration NC. It would be located in cn=sites,cn=configuration,dc=mycorp,dc=com in the mycorp.com forest. You can create new sites, subnets, and links, set replication schedules for each link, and so on.

Other replication-related tools are available in the Windows Support Tools:
RepAdmin

A command-line tool for administering replication.
ReplMon

A graphical utility for managing and monitoring replication.

5.2 Data Replication

Microsoft has introduced a number of new terms for Active Directory replication, and most of them will be completely unfamiliar to anyone new to Active Directory. To properly design replication, you need to understand how replication works, but more to the point, you need to understand how replication works using these new terms, which are used throughout both Microsoft's documentation and its management tools. Here is the list of the terms you'll encounter as we explain replication. These definitions will make more sense later.

Update Sequence Number (USN)

This 64-bit value, which is assigned to each object, increments every time a change takes place.

Originating write/update and replicated write/update

A change made to an object on a specific DC is an originating write; replication of that change to all other DCs is a replicated write.

High-Watermark Vector

This USN represents the maximum number of changes ever to occur on a particular NC.

Up-To-Date Vector

This is the USN on a specific server that represents the last originating write for an NC on that server.

Tombstone

Because of the complex replication available in Active Directory, simply deleting an object could result in it being re-created at the next replication interval, so deleted objects are tombstoned instead. This basically marks them as deleted. Objects marked as tombstoned are actually deleted 60 days after their original tombstone status setting; however, this time can be changed by modifying the tombstoneLifetime attribute of cn=DirectoryServices,cn=WindowsNT, cn=Services,cn=Configuration,dc=mycorp,dc=com for the mycorp.com forest.

Property version number

This number indicates how often this particular property has been updated.

Timestamp

This time and date are stored on an object for comparison checking.

Globally Unique Identifier (GUID)

This system-generated alphanumeric string represents a unique identifier for an object within an enterprise.

Flexible Single Master Operations (FSMO)

This term designates a server that performs one of the following roles: PDC Emulator, Infrastructure Master, RID Master, Schema Master, or Domain Naming Master.

5.2.1 A Background to Metadata—Data That Governs the Replication Process

Active Directory replication enables data transfer between NCs on different servers without ending up in a continuous replication loop or missing any data. To make this process work, each NC holds a number of pieces of information that specifically relate to replication within that particular NC. So the replication data for the Schema NC is held in the Schema NC and is separate from the replication data for the Configuration NC, which is held in the Configuration NC.

To minimize the use of abbreviations, we will refer to DCs from now on simply as servers. The terms property and attribute are also used interchangeably.

5.2.1.1 The High-Watermark Vector and orginating/replicated updates

5.3 Summary

We've now looked at the importance of the site topology in Active Directory and how that relates to your physical network. We've also considered the metadata that governs the replication process, how the system keeps track of changes to objects and properties automatically, how data is replicated among servers including propagation dampening, and how conflicts are reconciled.

Later on, in Chapter 9, we take this knowledge further and show you how Active Directory manages and automatically generates the replication connections that exist both within and between sites. With that knowledge, we can move on to the design principles for sites and links in Active Directory.

Chapter 6. Active Directory and DNS

One of the big advantages of Active Directory over its predecessor, Windows NT, is the reliance on the Domain Name System (DNS) as opposed to the Windows Internet Naming Service (WINS) for name resolution. DNS is the ubiquitous, standards-based naming service used on the Internet. WINS, on the other hand, never garnered industry support and, because it is a proprietary Microsoft offering, was typically used only to support Windows NT NOS environments.

The good news is that with Active Directory the dependencies on WINS have been eliminated, but the potentially bad news is that Active Directory has a lot of dependencies on the DNS infrastructure. It is only potentially bad based on the flexibility of your DNS environment. Often, the groups that manage DNS and Active Directory within an organization are different, and getting the two teams to agree on implementation can be difficult due to political turf battles or technology clashes.

The intent of this chapter is to provide you with a good understanding of how Active Directory uses DNS and a description of some of the options for setting it up within your organization. We will briefly touch on some DNS basics but will not go into much depth on how to configure and administer the Windows DNS server. For more information on those topics, we highly recommend DNS on Windows 2000 by Matt Larson and Cricket Liu (O'Reilly & Associates).

6.1 DNS Fundamentals

DNS is a hierarchical name resolution system. It is also the largest public directory service deployed. Virtually every company uses DNS for name resolution services, including hostname to IP address, IP address to hostname, and hostname to alternate hostname (aliases). DNS is a well-documented standard that has been around since the early days of the Internet. The RFCs in the following list cover some of the basics of DNS:

- RFC 1034, "Domain Names - Concepts and Facilities"
- RFC 1035, "Domain Names - Implementation and Specification"
- RFC 1912, "Common DNS Operational and Configuration Errors"
- RFC 1995, "Incremental Zone Transfer in DNS"
- RFC 1996, "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)"
- RFC 2181, "Clarifications to the DNS Specification"

There are three important DNS concepts that every Active Directory administrator must understand. Zones are delegated portions of the DNS namespace, resource records contain name resolution information, and dynamic DNS allows clients to add and delete resource records dynamically.

6.1.1 Zones

A zone is a collection of hierarchical domain names, the root of which has been delegated to one or more name servers. For example, let's say that the mycorp.com DNS namespace was delegated to ns1.mycorp.com. All domain names contained under mycorp.com that ns1.mycorp.com was authoritative for would be considered part of the mycorp.com zone. A subset of the mycorp.com zone could be delegated to another server, for example, subdomain1.mycorp.com, could be delegated to ns2.mycorp.com. At that point, subdomain1.mycorp.com becomes its own zone for which ns2.mycorp.com is authoritative.

The terms zone and domain are often confused in DNS parlance. A domain or domain name can actually be any type of name contained within a zone. The term zone has significance in relation to a portion of the namespace that has been delegated. A subdomain on one server may be a zone on another. The difference is determined by identifying the root of the contiguous namespace that was delegated.

6.1.2 Resource Records

A resource record is the unit of information in DNS. A zone is essentially a collection of resource records. There are various resource record types that define different types of name lookups. Table 6-1 lists some of the more common resource record types.

Table 6-1. Commonly used resource record types

Record type	Name	Description

6.2 DC Locator

One of the fundamental issues for clients in any NOS environment is finding the most optimal domain controller (DC) to authenticate against. The process under Windows NT was not very efficient and could cause clients to authenticate to domain controllers in the least optimal location. With Active Directory, clients use DNS to locate domain controllers via the DC locator process. To illustrate at a high level how the DC locator process works, we will describe an example where a client has moved from one location to another and needs to find a DC:

1. A client previously located in Site A logs in from Site B.
2. When the client boots up, it thinks it is still in Site A, so it proceeds to contact a DC in Site A using DNS unless the server name was previously cached.
3. The DC in Site A receives the request and realizes that the client should now be talking to a DC in Site B due to its IP address changing. If the server does not cover Site B, it will return the clients new site in the reply.
4. The client will then perform a DNS lookup to find a DC in Site B.
5. The client then contacts the DC in Site B. Three things can happen: the DC responds and authenticates the client; the DC fails to respond (it could be down), and the client attempts to use a different DC in Site B; or the DC fails to respond, and the client searches and fails to find another DC in Site B, instead turning back to the DC in Site A and authenticating with the original server.

The two main things that are needed to support the DC locator process are proper definition of the site topology in Active Directory and containment of all the necessary Active Directory related resource records in DNS. In the next section, we will describe the purpose of the resource records used in Active Directory. For a more detailed description of how the DC locator process works, including the specific resource records that are queried during the process, check out Microsoft Knowlede Base (KB) article 247811 "How Domain Controllers Are Located in Windows" and Microsoft KB article 314861 "How Domain Controllers Are Located in Windows XP" at http://support.microsoft.com.

6.3 Resource Records Used by Active Directory

When you promote a domain controller into a domain, a file containing the necessary resource records for it to function correctly within Active Directory is generated in *%SystemRoot%\System32\Config\netlogon.dns*.

The contents of the file will look something like the following for a DC named moose.mycorp.com in the mycorp.com domain with IP address 10.1.1.1. We've reordered the file a bit to group records of similar purpose together. Note that some lines may wrap due to their length.

```
 mycorp.com. 600 IN A 10.1.1.1
ec4caf62-31b2-4773-bcce-7b1e31c04d25._msdcs.mycorp.com. 600 IN CNAME moose.mycorp.
com.
gc._msdcs.mycorp.com. 600 IN A 10.1.1.1
_gc._tcp.mycorp.com. 600 IN SRV 0 100 3268 moose.mycorp.com.
_gc._tcp.Default-First-Site-Name._sites.mycorp.com. 600 IN SRV 0 100 3268 moose.
mycorp.com.
_ldap._tcp.gc._msdcs.mycorp.com. 600 IN SRV 0 100 3268 moose.mycorp.com.
_ldap._tcp.Default-First-Site-Name._sites.gc._msdcs.mycorp.com. 600 IN SRV 0 100 3268
moose.mycorp.com.
_kerberos._tcp.dc._msdcs.mycorp.com. 600 IN SRV 0 100 88 moose.mycorp.com.
_kerberos._tcp.Default-First-Site-Name._sites.dc._msdcs.mycorp.com. 600 IN SRV 0 100
88 moose.mycorp.com.
_kerberos._tcp.mycorp.com. 600 IN SRV 0 100 88 moose.mycorp.com.
_kerberos._tcp.Default-First-Site-Name._sites.mycorp.com. 600 IN SRV 0 100 88 moose.
mycorp.com.
_kerberos._udp.mycorp.com. 600 IN SRV 0 100 88 moose.mycorp.com.
_kpasswd._tcp.mycorp.com. 600 IN SRV 0 100 464 moose.mycorp.com.
_kpasswd._udp.mycorp.com. 600 IN SRV 0 100 464 moose.mycorp.com.
_ldap._tcp.mycorp.com. 600 IN SRV 0 100 389 moose.mycorp.com.
_ldap._tcp.Default-First-Site-Name._sites.mycorp.com. 600 IN SRV 0 100 389 moose.
mycorp.com.
_ldap._tcp.pdc._msdcs.mycorp.com. 600 IN SRV 0 100 389 moose.mycorp.com.
_ldap._tcp.97526bc9-adf7-4ec8-a096-0dbb34a17052.domains._msdcs.mycorp.com. 600 IN SRV
0 100 389 moose.mycorp.com.
_ldap._tcp.dc._msdcs.mycorp.com. 600 IN SRV 0 100 389 moose.mycorp.com.
_ldap._tcp.Default-First-Site-Name._sites.dc._msdcs.mycorp.com. 600 IN SRV 0 100 389
moose.mycorp.com.
```

While it may look complicated, it isn't. Let's go through what these records actually mean, splitting the records up into sections for ease of understanding. To start with, the first record is for the domain itself:

```
 mycorp.com. 600 IN A 10.1.1.1
```

Each DC attempts to register an A record for its IP address for the domain it is in. A quick and easy way to get a list of all the domain controllers in a domain is to simply look up the A record for the domain name. We will now walk through that query to show the domain controllers that have registered an A record for the mycorp.com domain:

```
 > nslookup mycorp.com
Server:  moose.mycorp.com
Address:  10.1.1.1

Name:    mycorp.com
Addresses:  10.1.1.1, 10.1.1.2, 10.1.1.3

> nslookup 10.1.1.1
Server:  moose.mycorp.com
Address:  10.1.1.1

Name:    moose.mycorp.com
Addresses:  10.1.1.1

> nslookup 10.1.1.2
Server:  moose.mycorp.com
Address:  10.1.1.1

Name:    deer.mycorp.com
```

6.4 Delegation Options

Now that we've covered what Active Directory uses DNS for, we will review some of the options for setting up who is authoritative for the Active Directory-related zones. Ultimately, the decision boils down to whether you want to use your existing DNS servers or different servers, such as the domain controllers, to be authoritative for the zones. There are many factors that can affect this decision, including:

- Political turf battles between the AD and DNS teams
- Initial setup and configuration of the zones
- Support and maintenance of the zones
- Integration issues with existing administration software and practices

We will look at each of these factors as they apply to delegating the AD zones. Other slight variations of these options do exist, but we will discuss only the basic cases.

6.4.1 Not Delegating the AD DNS Zones

The first impulse of any cost-conscious organization should be to determine whether the existing DNS servers can be authoritative for the AD zones. That would entail populating all the necessary resource records required by each DC. While this sounds fairly trivial, there are several issues to be aware of.

6.4.1.1 Political factors

By utilizing the existing DNS servers for the AD DNS zones, the AD administrators will likely not have the same level of control as they would if the zones were delegated and managed by them. While it does limit the scope of control for a crucial service used by Active Directory, some AD administrators may find it a blessing!

6.4.1.2 Initial setup and configuration

The initial population of the AD resource records can be burdensome depending on how you manage your resource records and how easy it will be for you to inject new ones. The domain controllers try to register their resource records via DDNS on a periodic basis. Most organizations do not allow just any client to make DDNS updates due to the potential security risks. For that reason, you'll need to configure your existing DNS servers to allow the domain controllers to perform DDNS updates. And unless you restrict which zones the domain controllers can send DDNS updates for, it opens a potential security hole. If a domain controller can update any zone, an AD administrator could conceivably perform individual updates for any record in any zone while logged onto that DC. This should not typically be a problem, but depending on how paranoid the DNS administrators are, it could be a point of contention.

6.4.1.3 Support and maintenance

Assuming the existing DNS servers are stable and well supported (as they tend to be in most organizations), name resolution issues should not be a problem for AD DCs or other clients that are attempting to locate a DC via DNS. Ongoing maintenance of the DC resource records can be an issue, as pointed out previously. Each time you promote a new DC in the forest, you'll need to make sure it is allowed to register all of its records via DDNS. The registration of these records could be done manually, but due to the dynamic nature of the AD resource records, they would have to be updated on a very frequent basis (potentially multiple times a day). Yet another option is to programmatically retrieve the netlogon.dns file from each domain controller on a periodic basis and perform the DDNS updates from a script. In large environments, the manual solution will probably not scale, and either DDNS or a programmatic solution will need to be explored.

6.5 Active Directory Integrated DNS

If you've decided to host the AD DNS zones on your domain controllers, you should strongly consider using AD integrated zones. This section will explain some of the benefits of using AD integrated DNS versus standard primary zones.

In the normal world of DNS, you have two types of name servers: primary and secondary (a.k.a. slaves). The primary name server for a zone holds the data for the zone in a file on the host and reads the entries from there. Each zone typically has only one primary. A secondary gets the contents of its zone from the primary that is authoritative for the zone. Each primary name server can have multiple secondary name servers. When a secondary starts up, it contacts its primary and requests a copy of the relevant zone via zone transfer. The contents of the secondary file are then dynamically updated over time according to a set scheme. This is normally a periodic update or triggered automatically by a message from the primary stating that it has received an update. This is a very simplified picture, as each name server can host multiple zones, allowing each server to have a primary role for some zones and a secondary for others.

Each type of server can resolve name queries that come in. However, if a change must be made to the underlying contents of the DNS file, it has to be made on the primary name server for that zone. Secondary name servers cannot accept updates.[1]

[1] This isn't strictly true. While slaves cannot process updates, they can and do forward updates that they receive to the primary name server.

Another option available with Active Directory and Windows DNS server is to integrate your DNS data into Active Directory. Effectively, this means that you can store the contents of the zone file in Active Directory as a hierarchical structure. Integrating DNS into Active Directory means that the DNS structure is replicated among all DCs of a domain. Each DC holds a writeable copy of the DNS data. The DNS objects stored in Active Directory could be updated on any DC via LDAP operations or through DDNS against DCs that are acting as DNS servers. This effectively makes the entire set of DCs act like primary name servers, where each DC can write to the zone and issue authoritative answers for the zone. This is a far cry from the standard model of one primary name server and one or more secondary name servers, which has the obvious downside of a single point of failure for updates to DNS.

6.5.1 Replication Impact

While AD Integrated DNS has many advantages, the one potential drawback is how DNS data gets replicated in Active Directory. Under Windows 2000, AD Integrated zones are stored in the System container for a domain. That means that every domain controller in that domain will replicate that zone data regardless of whether the domain controller is a DNS server. For domain controllers that are not DNS servers, there is no benefit to replicating the data. Fortunately, there is a better alternative in Windows Server 2003, using application partitions as described in the next section.

6.6 Using Application Partitions for DNS

Application partitions, as described in Chapter 3, are user-defined partitions that have customized replication scope. Domain controllers that are configured to contain replicas of an application partition will be the only servers that replicate the data contained within the partition. One of the benefits of application partitions is that they are not limited by domain boundaries. You can configure domain controllers in completely different domains to replicate an application partition. It is for these reasons that application partitions make a lot of sense for storing AD Integrated DNS zones. No longer do you have to store DNS data within the domain context and replicate to every domain controller in the domain, even if only a handful are DNS servers. With application partitions you can configure Active Directory to replicate only the DNS data between the domain controllers running the DNS service within a domain or forest.

When installing a new Windows Server 2003 Active Directory forest, the default DNS application partitions are created automatically. If you are upgrading from Windows 2000, you can manually create them by using the DNS MMC snap-in or the dnscmd.exe utility. There is one default application partition for each domain and forest. When configuring an AD Integrated zone in a Windows Server 2003 forest, you have several options for storing the DNS data. These options are listed in Table 6-2.

Table 6-2. Active Directory Integrated DNS zone storage options

Distinguished name	**Replication scope**
cn=System,*DomainDN* Example: cn=System,dc=amer,dc=mycorp,dc=com	To all domain controllers in the domain. This is the only storage method available under Windows 2000.
dc=domaindnszones,*DomainDN* Example: dc=domaindnszones,dc=amer, dc=mycorp,dc=com	To domain controllers in the domain that are also DNS servers.
dc=forestdnszones,*ForestDN* Example: dc=forestdnszones,dc=mycorp,dc=com	To domain controllers in the forest that are also DNS servers.
AppPartitionDN Example: dc=dnsdata,dc=mycorp,dc=com	To domain controllers that have been configured to replicate the application partition.

6.7 Summary

Active Directory relies heavily on DNS. In fact, Microsoft has shifted completely away from WINS for name resolution within the NOS in favor of standards-based DNS. The DC locator process is a core DNS-based function used within Active Directory to help domain controllers and clients locate domain controllers that have certain properties, such as residing in a particular site or being a Global Catalog server or PDC emulator. Deciding how to manage the AD DNS zones can be a difficult decision, with each option having its own advantages and disadvantages. If you delegate the zones to domain controllers, AD Integrated zones can save a lot of time in maintenance and upkeep. In Windows Server 2003, you can use application partitions to replicate AD Integrated zones to only the domain controllers that are acting as DNS servers. This can greatly reduce replication traffic in some situations compared to Windows 2000 Active Directory, which replicated DNS data to every domain controller in a domain regardless of whether it was a DNS server.

Chapter 7. Profiles and Group Policy Primer

Profiles and group policies are large topics, and they are worth treating properly so that you get the most from them in your environment. The goal of policy-based administration is for an administrator to define the environment for users and computers once, then rely on the system to enforce that state. Under Windows NT, this could be very challenging, but with Active Directory group policies, this capability is much more readily available. This chapter is the introduction to the subject, and Chapter 10 builds on it to show how policies work in Active Directory, how to design an OU structure to incorporate them effectively, and how to manage them with the Group Policy Management Console, a new MMC snap-in available for Windows Server 2003 Active Directory.

In Windows NT, system policies had a number of limitations. System policies:

- Were set at the domain level
- Were not secure
- Could only apply to users, groups of users, or computers
- Tended to set values until another policy specifically unset them
- Were limited to desktop lockdown

The scope and functionality of Active Directory group policies is much greater than system policies. Group policies:

- Can be applied to individual clients, sites, domains, and Organizational Units
- Are highly secure
- Can apply to users, computers, or groups of either
- Can set values and automatically unset them in specified situations
- Can do far more than just a desktop lockdown

With group policies, an administrator can define a large number of detailed settings to be enforced on users throughout the organization, and he can be confident that the system will take care of things. Let's take a simple example from Leicester University. Administrators wanted the Systems Administrator toolset, which normally is installed only on servers, to be available on workstations also. While they could install these tools on their own PCs, they actually wanted the tools to follow them around the network and be available from any PC that they chose to log on from. However, they didn't want to leave these tools installed on that PC when they logged off. Prior to Active Directory, the admins would have had to arrive at a client, log on, install the toolset, do whatever was required at a client, uninstall the toolset, and finally log off. This would be a considerable chore for a large number of machines. Active Directory group policies can be used to specify that the toolset is to be automatically installed on any client that an administrator logs on to. That way, an administrator could go straight to the Start menu and find the tools available. After logging off, the same group policy would uninstall the toolset from the machine.

7.1 A Profile Primer

Profiles and group policies are tightly related, but they serve completely different functions. To make things clear, we'll cover the essentials of profiles so that you can understand how to manipulate them using group policies.

Let's consider a Windows XP workstation with a newly created account for a user named Richard Lang with the username RLang. When Richard logs on to the client, the system creates a profile directory for him, corresponding to his username, in the Documents and Settings directory. If Richard were to log on to a Windows NT workstation or to a Windows 2000 workstation that was upgraded from a previous version of Windows NT, the profile would be created under the *%systemroot%\Profiles*[2] directory. On a fresh Windows 2000 install or Windows XP, the profiles are stored under *%systemdrive%\Documentand Settings*.

[2] %systemroot% is the system environment variable that refers to the location of the Windows operating system files. If Windows NT were installed on drive C: in the normal way, %systemroot% would be C:\WINNT. The %systemdrive% variable contains the drive letter of the drive the operating system was installed on.

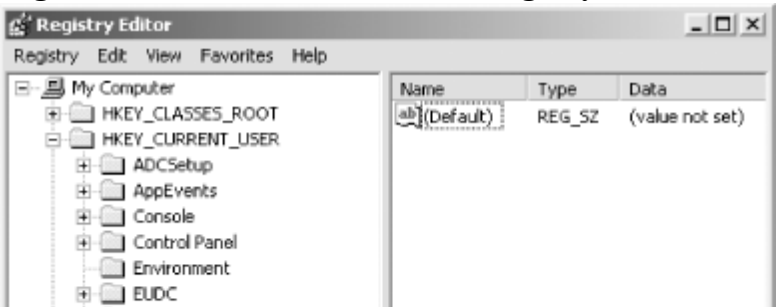
Inside this directory, the system places a file called *NTUSER.DAT*, along with various other data files. Let's concentrate on the *NTUSER.DAT* file for a moment. This file contains what is known generally as the user portion of the registry. All Windows-based operating systems have a registry that consists of two parts: the so-called user portion represented by the file *NTUSER.DAT* (or *USER.DAT* on Windows 9x systems) and the system or computer portion of the registry, which is stored in *%systemroot%\system32\config*. The user part of the registry holds information indicating what screensaver should be used for that user; what colors, background, and event sounds are set; where the user's My Documents folder points to; and so on. The system portion of the registry holds hardware device settings, installed software information, and so on. When a user logs on to a client, the combined effects of the settings for the machine held in the system portion of the registry and the settings for the user held in the user portion of the registry take effect.

When you use a tool such as *REGEDIT.EXE* or *REGEDT32.EXE* to examine the registry on a machine, both portions of the registry are opened and displayed together for you to look at within one tool.

The two registry tools were developed with different requirements in mind, but with Windows Server 2003 they have been merged. The REGEDIT tool was developed initially for Windows 9x clients and thus allows for management of the datatypes as well as for rapid searching for any key or value that contains a given word or phrase. REGEDT32, on the other hand, was designed to support the extra datatypes present in Windows NT and Windows 2000. However, REGEDT32 had an awful search mechanism that allowed searches only through keys. In Windows Server 2003, REGEDIT was updated to support many of the features present in REGDT32. Now if you run REGEDT32, you will bring up the REGEDIT interface.

Figure 7-1 shows a view of the registry on a Windows 2000 client when viewed from REGEDIT. The screenshot also shows the five registry hives (as they are known) available to Windows 2000. The two important hives are HKEY_LOCAL_ MACHINE, also known as HKLM, which corresponds to the system part of the registry, and HKEY_CURRENT_USER, also known as HKCU, which corresponds to the user portion of the registry.

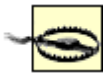
Figure 7-1. A REGEDIT view of the registry on a Windows 2000 Professional client



7.2 Capabilities of GPOs

GPOs can be edited using the Group Policy Object Editor (GPOE), formerly theGroup Policy Editor (GPE), which is an MMC snap-in. The GPOE is limited to managing a single GPO at a time and cannot be used to link a GPO. For this reason, Microsoft developed the Group Policy Management Console (GPMC) MMC snap-in, which was released around the same time as Windows Server 2003, as a web download from http://download.microsoft.com. The GPMC provides a single interface to manage all aspects of GPOs, including editing (through the GPOE), viewing the resultant set of policies (RSOP), and linking to domains, sites, and OUs. We will cover these tools in much more detail in Chapter 10.

Most settings in a GPO have three states: enabled, disabled, and unconfigured. By default, all settings in a GPO are unconfigured. Any unconfigured settings are ignored during application, so the GPO comes into play only when settings have actually been configured. Each setting needs to be configured as enabled or disabled before it can be used, and in some cases the option needs no other parameters. In other cases, a host of information must be entered to configure the option; it all depends on what the option itself does.



Enabling and disabling most options is fairly straightforward. However, due to Microsoft's choice for the names of certain settings for GPOs, you actually can have the choice of enabling or disabling options with names like "Disable Access to This Option". By default, this setting isn't in use, but you can disable the disable option (i.e., enable the option) or enable the disable option (i.e., disable the option). Be careful and make sure you know which way the setting is applied before you actually go through with the change.

GPOs can apply a very large number of changes to computers and users that are in Active Directory. These changes are grouped together within the GPOE under the three headings of Software Settings, Windows Settings, and Administrative Templates. There are two sets of these headings, one under Computer Configuration and one under User Configuration. The items under the three headings differ, as the settings that apply to users and to computers are not the same.

Some of the settings under Administrative Templates would look more sensible under the other two sections. However, the Administrative Templates section holds data that is entirely generated from the Administrative Template (ADM) files in the system volume; so it makes more sense to include all the ADM data together. ADM files contain the entire set of options available for each setting, including explanations that are shown on the various property pages in the GPOE.

ADM files can be added and removed by right-clicking either Administrative Template location in the GPOE and choosing Add/Remove Templates. Very comprehensive information on customizing GPOs and adding in your own templates can be found in Microsoft's Windows 2000 Group Policy technical white paper. Check out the following URL for more information:

http://www.microsoft.com/windows2000/techinfo/howitworks/management/grouppolwp.asp

In Windows Server 2003 Active Directory, Microsoft extended the capabilities of GPOs significantly. Over 160 new settings have been added, some of which cover new areas, such as the netlogon process, DNS configuration, networking QOS and wireless, and terminal services. We'll now give an overview of the main categories of settings available with GPOs and provide a brief explanation for some of the main capabilities of each.

7.2.1 Software Installation Settings (Computer and User)

7.3 Summary

Whew! That's a lot of settings. Hopefully we've given you a good idea of just how powerful GPOs are in Active Directory. We've now covered the basics of what profiles can do and how modifications to a centralized profile make a lot of sense and are easy to manage. We've also taken a very in-depth look at the diverse sort of registry, user interface, file permission, and system changes that can be made using GPOs. In Chapter 10, we'll cover how to design and manage your GPOs.

This concludes our initial introduction to Active Directory. In Part II, we will dive into some of the important issues around designing and maintaining an Active Directory environment.

Part II: Designing an Active Directory Infrastructure

You should start your Active Directory design with the namespace. However, you will not be able to complete the logical namespace design until you have the physical design sketched out. It's very much a chicken-and-egg situation. You should plan to go through and complete a rough draft of the namespace design, then make a rough draft of the physical design, then consider modifications to both.

Next you can consider the Group Policy Object (GPO) design. Group Policy Objects control such things as user-environment lockdown, forced registry changes, application availability, and so on, to sets of machines or users. Because these relate to sites, domains, Organizational Units, users, computers, and groups in your Active Directory, it makes sense in my experience to incorporate these changes into a namespace and site design that already exist.

You can then take a look at security and at tailoring Active Directory to your own requirements by modifications to the Schema. Finally, this section takes a brief look at the present and the future of integrating and interoperating Active Directory with other directories and operating systems and of migrating to Active Directory.

Chapter 8. Designing the Namespace

The basic emphasis of this chapter is on reducing the number of domains that you require for Active Directory while gaining administrative control over sections of the namespace using Organizational Units. This chapter aims to help you create a domain namespace design. That includes all the domains you will need, the forest and domain-tree hierarchies, and the contents of those domains in terms of Organizational Units and even groups.

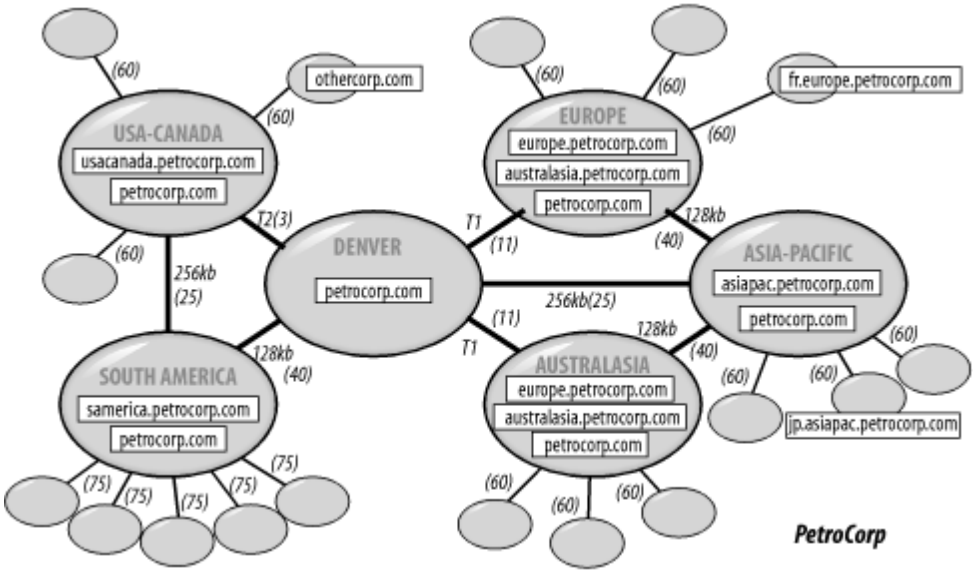
There are a number of restrictions that you have to be aware of when beginning your Active Directory design. We will introduce you to them in context as we go along, but here are some important ones:

- Too many Group Policy Objects (GPOs) means a long logon time as the group policies are applied to sites, domains, and Organizational Units. This obviously has a bearing on your Organizational Unit structure, as a 10-deep Organizational Unit tree with GPOs applying at each branch will incur more GPO processing than a 5-deep Organizational Unit tree with GPOs at each branch.
- Under Windows 2000, you cannot rename a domain once it has been created. Fortunately, with Windows Server 2003, this limitation has been removed, although the rename process is tedious. You can even rename forest root domains once you've reached the Windows Server 2003 forest functional level.
- You can never remove the forest root domain without destroying the whole forest in the process. The forest root domain is the cornerstone of your forest.
- The Schema Admins and Enterprise Admins groups exist in the forest root domain only. So if you are migrating from a previous version of NT, be cognizant of the fact that the administrators of the first domain you migrate can have control over these groups and over Active Directory.
- Lack of a regional catalog is problematic. Imagine that you have 20 printers in your office in Sweden and 12 printers in your office in Brazil. The users in Sweden will never need to print to the printers in Brazil, and the users in Brazil will never need to print to the printers in Sweden. However, by default, details of all printers are published in the GC. Thus, whenever changes are made to printers in Sweden, all the changes get replicated to the GCs on the Brazil servers because the GC replicates all of its data everywhere. You have three options. You can decide not to replicate any printer data and force printer seraches to hit Active Directory each time, you can replicate all printer data everywhere, or you can create an application partition to host printer data and replicate it to designated domain controllers.
- Multiple domains cannot be hosted on a single DC. Imagine 3 domains off a root located in the United States, which correspond to 3 business units. Now imagine a small office of 15 people in Eastern Europe or Latin America with a slow link to the main site. The 15 users are made up of 3 sets of 5; each set of 5 users uses one of the 3 business units/domains. If you as an administrator decide that the slow link is too slow and you would like to put in a DC for the 3 domains at the local server and to ease replication, the small office will have to install 3 DCs.

8.1 The Complexities of a Design

Active Directory is a complex beast, and designing for it isn't easy. Take a look at a fictitious global company called PetroCorp, depicted in Figure 8-1.

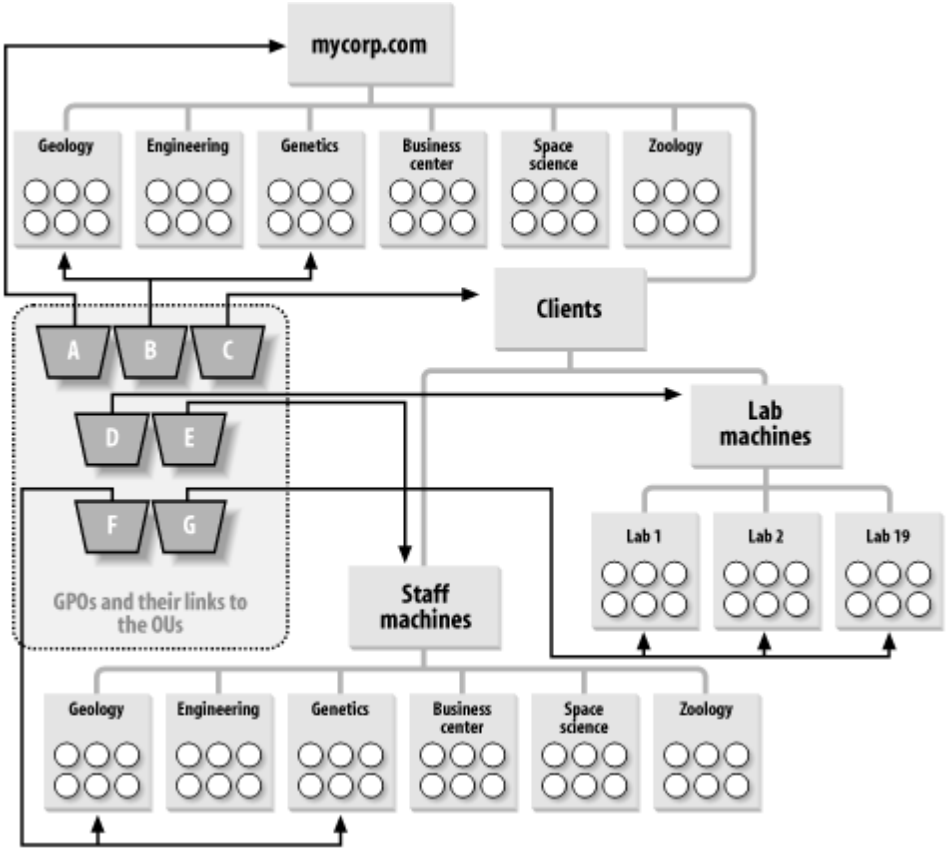
Figure 8-1. The sites and servers of a company called PetroCorp



Here you can see a huge network of sites linked with various network connections across wide area networks. A variety of domains seems to exist for *othercorp.com* and *petrocorp.com*, and as each one of those square boxes represents a single domain controller (the servers that host Active Directory in an organization), you can see that some of the servers will need to replicate data across those WAN links. *petrocorp.com*, for example, seems to need to replicate to all the major sites, since it has domain controllers (DCs) in each of those sites.

Take a look at Figure 8-2, which shows a much more complex hierarchy.

Figure 8-2. A complex domain tree showing GPOs



It's possible to see the users and computers in all the Organizational Units in this view, and the structure seems to be set up so that Group Policy Objects (GPOs, represented by trapezoids) can act on various portions of the tree. These

8.2 Where to Start

Before you sit down to make your design, you will need to obtain some important pieces of information. You will need a copy of your organizational structure. This is effectively the document that explains how your organization's business units fit together in the hierarchy. Next you will need a copy of the geographical layout of your company. This includes the large-scale picture in continents and countries and also the individual states, counties, or areas in which you have business units. Third, you will need a copy of the network diagram, indicating the speeds of connection between the various sites. Finally, you need a copy of any diagrams and information on any systems that will need to interface to Active Directory, such as existing X.500 directories, so that you can take them into account. Once you've gathered the information, you can sit down and plan your design.

8.3 Overview of the Design Process

The namespace design process takes place in three stages:

Design of the domain namespace

During the first stage, you deal with the namespace design itself. That means calculating the number of domains you need, designing the forest and tree structure, and defining the naming scheme for workstations, servers, and the network as a whole.

Design of the internal domain structure

During the second stage, you need to concentrate on the internal structure of each domain that you have previously noted. Here you also need to use your business model as a template for the internal structure and then move on to consider how administration and other rights will be delegated. The internal structure can also be modified depending on how you intend to use Group Policy Objects; this will be covered in Chapter 10.

Global catalog design

During the third stage, you work out your designs for the global catalog (GC).

When you are finished with your design, you can implement the design by setting up a test forest in a lab environment. This will enable you to get a better feel for how the design actually works and whether there is anything you have failed to consider. We can't stress enough the use of a test environment.

8.4 Domain Namespace Design

The first stage in your design is to work out the domain, domain-tree, and forest configuration of your network. The best way to do this is to make a first pass at designing the domains and then structure them together into a series of trees. Before we start, however, let's take a look at our objectives for this part of the design.

8.4.1 Objectives

There are two objectives for the design of the domain namespace:

- Designing Active Directory to represent the structure of your business
- Minimizing the number of domains by making much more use of the more flexible Organizational Units

8.4.1.1 Represent the structure of your business

You need to make Active Directory look as much like your business structure, geographical or organizational, as possible. With geographical structure, your business runs itself as self-contained units within each geographical site. In this model, people at those sites handle administration for each site. Under the organizational or political model, the business is based on a series of departments that have members from a number of different geographical sites. Normally, with this structure, the organization has a head office for all departments at one location, but that is not always the case.

In the former model, finance units based in France and Australia would be separate finance departments. In the latter model, France and Australia would have geographical finance branches of a larger finance department controlled from a head office.

It doesn't matter to Active Directory which model you choose, except that the intention is to mirror the structure of your business in the Active Directory design. If your business crosses both of these boundaries, it becomes less clear-cut. To make your design simpler to understand, you should choose to go with one model or the other. We would not suggest a mix-and-match approach unless you can definitely rationalize it, adequately represent it on paper, and delegate administration effectively.

If you already have a large investment in a TCP/IP infrastructure with organization or geographic-centered DNS zones, or you if have a large existing Exchange organization, you can use this as the basis of your design. Simply stated, if your DNS or Exchange setup is based on one model, go with that model for your Active Directory design. It should be obvious that it will be easier for an administrator to think about both areas if the designs are based on the same model.

8.4.1.2 Minimize the number of domains

Remember that implementing Active Directory presents an opportunity to reduce the number of domains you support. Each forest can store tens of millions of objects, which is more than enough for all the users, groups, and computers in most organizations. So size isn't a consideration. Each domain can also be partitioned using Organizational Units, allowing you to delegate different administrators for each Organizational Unit in a domain if you so desire. You do not have to create a new domain if you wish to delegate administration over a part of the system. These two aspects of Active Directory tend to eliminate a number of sizing and permission problems associated with traditional NT installations.

If you're an experienced NT domain designer, start trying to push from your mind the tendency to create multiple domains. Think in terms of multiple Organizational Units instead.

8.5 Design of the Internal Domain Structure

Having designed the domain namespace, you can now concentrate on the internals of each domain. The design process itself is the same for each domain, but the order is mostly up to you. The first domain that you should design is the forest root domain. After that, iterate through the tree, designing subdomains within that first tree. Once the tree is finished, go on to the next tree and start at the root as before.

In a tree with three subdomains called Finance, Sales, and Marketing under the root, you could either design the entire tree below Finance, then the entire tree below Sales, and so on, or you could design the three tier-two domains first, then do all the subdomains immediately below these three, and so on.

When designing the internals of a domain, you need to consider both the hierarchical structure of Organizational Units and the users and groups that will sit in those Organizational Units. Let's look at each of those in turn.

When we refer to a hierarchy, a tree, or the directory tree, we mean the hierarchical Organizational Unit structure within a domain. We are not referring to the hierarchy of domain trees in a forest.

8.5.1 Step 4—Design the Hierarchy of Organizational Units

Earlier, when we discussed how to design domains, we spoke of how to minimize the number of domains you have. The idea is to represent most of your requirements for a hierarchical set of administrative permissions using Organizational Units instead.

Organizational Units are the best way to structure your data because of their flexibility. They can be renamed and easily moved around within and between domains and placed at any point in the hierarchy without affecting their contents. These two facts make them very easy for administrators to manage.

There are four main reasons to structure your data in an effective hierarchy:
To represent your business model to ease management

Partitioning your data into an Organizational Unit structure that you will instantly recognize makes managing it much more comfortable than with every user and computer in one Organizational Unit.
To delegate administration

Active Directory allows you to set up a hierarchical administration structure that wasn't possible with Windows NT. If you have three branches, and the main administrator wants to make one branch completely autonomous with its own administrator but wants to continue to maintain control over the other two branches, it's easy to set up. In a way, most of the limitations that you come up against when structuring Active Directory are limits that you set: political necessities, organizational models, and so on. Active Directory really won't care how you structure your data.
To replace Windows NT resource domains

If you have a previous Windows NT installation with a master or multimaster domain model, you can replace your resource domains with Organizational Units in a single domain. This allows you to retain all the benefits of having resource domains (i.e., resource administration by local administrators who do not have account administration rights) without forcing you to have multiple domains that you don't really want or need.
To apply policies to subsets of your users and computers

As policies can be applied to each individual Organizational Unit in the hierarchy, you can specify that different computers and users get different policies depending on where you place them in the tree. For example, let's say that you want to place an interactive touch-screen client in the lobby of your headquarters and allow people to interact with whatever applications you specify, such as company reports, maps of the building, and so on. Locking this down in Windows NT (so that the client could not compromise your network in any way) required time and may have required that the client be in a separate domain or even standalone. With Active Directory, if you lock down a certain

8.6 Other Design Considerations

In many cases you may need to revise your namespace designs a number of times. Certainly GPOs will make a difference as to how you structure your users and computer objects, so we do not assume that one pass through a design process will be enough.

Once you have a basic design, there is nothing stopping you from putting that design to one side and working on identifying a perfect design for your Active Directory network, one that you would like to implement in your organization, ignoring all Active Directory-imposed design constraints. You then can work out how difficult it will be to move to that perfect design from the practical one that you worked out using the preceding steps. You can look at the feasibility of the move from one to the other and then rationalize and adjust your final design to take into account the factors you have listed. You can then use this as an iteration tool so that your final design is much closer to the perfection you are aiming for.

Apart from GPOs, which we cover in Chapter 7 and Chapter 10, there are other aspects of Active Directory design that we have not and will not be covering. For example, you are quite likely to want printers advertised in Active Directory so that they can be accessed easily using a simple search of Active Directory (which the Add Printer wizard now uses as the default option). You may want shares advertised in Active Directory, so that users can easily locate data partitions on a site nearest to them. The Distributed Filing System (DFS) that allows you to organize disjointed and distributed shares into a single contiguous hierarchy is a fine example of this in action. When you reference a share held by the DFS, the DFS uses Active Directory to automatically redirect your request to the closest share replica. There is also the matter of designing your own objects and attributes that you want to include. However, there are two points that you should consider:

- As a general rule, Active Directory should hold only static or relatively static data. At the very least, the lifetime of the data has to be greater than the time to replicate to all DCs throughout the organization. When considering which objects to add, don't consider adding objects with very short life spans.
- Any object that you include will have attributes that are held in the GC. For every type of object that you seek to store in Active Directory, check the schema class entry for that object to find out what attributes will be stored in the GC. Consider whether you need to add or remove items from that list by referring back to the design principles.

8.7 Design Examples

Having covered the design of the namespace, some real-world example designs are in order. We have created three fictitious companies that will serve as good models for demonstrations of the design process. We will also use these three companies in the following chapters. The companies themselves are not fully detailed here, although there is enough information to enable you to make a reasonable attempt at a namespace design. In the chapters that follow, we will expand the relevant information on each company as required for that part of the design.

We used a number of criteria to create these companies:

- The companies were set up to represent various organizations and structures.
- While each corporation has a large number of users and machines, the design principles will scale down to smaller organizations well.
- In these example corporations, we are not interested in how many servers each company has or where those servers are. These facts come into play in the next chapter on sites. We are interested in users, groups, machines, domains, and the business and administration models that are used.

8.7.1 TwoSiteCorp

TwoSiteCorp is an organization that employs 50,000 people using 50,000 machines. The organization spans 2 sites connected with a 128 Kb dedicated link. The London site has 40,000 clients and 40,000 employees, while the new expansion at the Leicester site has 10,000 clients and 10,000 employees. TwoSiteCorp's business model is based on a structure in which users are members of one of three divisions: U.K. Private Sector, U.K. Public Sector, and Foreign. No division is based entirely at one site. Various other minor divisions exist beneath these as required for the management structure. Administration is handled centrally from the major London site by a team of dedicated systems administrators.

8.7.1.1 Step 1—Set the number of domains

While TwoSiteCorp's 128 Kb link between its two physical locations is slow for site purposes, there is no need to split the two sites into two domains. No particular part of the organization has a unique policy requirement, because the administrators decided that they will implement one set of policies for all users. Finally, the sites already have two Windows NT domains installed. However, management has no desire to maintain either, so both will be rationalized into one domain. Thus, TwoSiteCorp will end up with one domain.

8.7.1.2 Step 2—Design and name the tree structure

TwoSiteCorp's single domain will be the forest root domain. The designers decide to name the domain *twositecorp.com* after their DNS domain name. With only one domain, they do not have to worry about any other trees or forests or the domain hierarchy.

8.7.1.3 Step 3—Design the workstation and server naming scheme

TwoSiteCorp decides that each machine name will be made up of four strings concatenated together. The first string is three characters representing the location of the machine (e.g., LEI or LON). The next three characters are used to indicate the operating system (e.g., WXP, W2K, NT4, or W98). The next string holds two or three letters indicating the type of machine (e.g., DC, SRV, or WKS). Finally, the last string is a six-digit numeric string that starts with 000001 and continues to 999999. The following are example machine names:

- LEIW2KDC000001

8.8 Designing for the Real World

It's very easy to get bogged down in the early stages of the namespace design without actually progressing much further. The stumbling block seems to be that it feels conceptually wrong to have only one domain, yet administrators can't put their finger on what the problem is. Experienced Windows NT administrators who manage multiple domains seem to find this much more of a problem than those coming from another operating system.

If you follow the guidelines in the initial steps of the namespace design, you quite probably will end up with one domain to start with. That's the whole point of the design process: to reduce the number of domains you need. Yet NT administrators tend to feel that they have conceptually lost something very important; with only one domain, somehow this design doesn't "feel right."

This is partly a conceptual problem: a set of domains with individual objects managed by different teams can feel more secure and complete than a set of Organizational Units in a single domain containing individual objects managed by different teams. It's also partly an organizational problem and, possibly, a political problem. Putting in an Active Directory environment is a significant undertaking for an organization and shouldn't be taken lightly. This change is likely to impact everyone across the company, assuming you're deploying across the enterprise. Changes at that level are likely to require ratification by a person or group who may not be directly involved on a day-to-day basis with the team proposing the change. So you have to present a business case that explains the benefits of moving to Active Directory.

8.8.1 Identify the Number of Domains

Following our advice in this chapter and Microsoft's official guidelines from the white papers or Resource Kit will lead most companies to a single domain for their namespace design. It is your network, and you can do what you want. More domains give you better control over replication traffic but may mean more expense in terms of hardware. If you do decide to have multiple domains but have users in certain locations that need to log on to more than one domain, you need DCs for each domain that the users need in that location. This can be expensive. We'll come back to this again later, but let's start by considering the number of domains you need.

If the algorithm we use to help you determine the number of domains gives you too small a figure in your opinion, here's how you can raise it:

- Have one domain for every single-master and multimaster Windows NT domain that you have. If you are using the Windows NT multimaster domain model, consider the entire set of multimasters as one domain under Active Directory (use Organizational Units for your resource domains).
- Have one domain per geographical region, such as Asia-Pacific, Africa, Europe, and so on.
- Have extra domains whenever putting data into one domain would deny you the control over replication that you would like if you used Organizational Units instead. It's all very well for us to say that Organizational Units are better, but that isn't true in all situations. If you work through the algorithm and come up with a single domain holding five Organizational Units, but you don't want any of the replication traffic from any of those Organizational Units to go around to certain parts of your network, you need to consider separate domains.

Even Microsoft didn't end up with one domain. They did manage to collapse a lot of Windows NT domains, though, and that's what you should be aiming for if you have multiple Windows NT domains.

8.8.2 Design to Help Business Plans and Budget Proposals

There are two parts to this: how you construct a business case itself for such a wide-reaching change and how you can show that you're aiming to save money with this new plan.

8.9 Summary

In this chapter, we presented a series of seven steps toward effective namespace design:

1. Decide on the number of domains.
2. Design and name the tree structure.
3. Design the workstation and server naming scheme.
4. Design the hierarchy of Organizational Units.
5. Design the users and groups.
6. Design the Global Catalog.
7. Design the application partition structure.

Following these seven steps allows you to solve the two main objectives of this chapter:

- Come up with an Active Directory namespace design to represent the structure of your business.
- Minimize the number of domains by making much more use of the more flexible Organizational Units.

While we've shown you how to start to design your Active Directory, there is still a long way to go. Designing the namespace of domains, trees, and forests and the internal Organizational Unit hierarchy according to the guidelines given here means that you should have a structural template that represents your business model within the preceding restrictions. Hopefully this design makes sense in terms of your organization and will be simpler to manage.

The rest of the design still needs to be completed. You need to look at the low-level network links between sites and how they will affect your replication decisions. You then need to tackle the subject of how to revise the initial namespace design based on Group Policy Objects, security delegation and auditing, schema changes, and so on. Next we'll move on to designing the physical site topology that the DCs use when communicating with one another.

Chapter 9. Creating a Site Topology

As we mentioned in Chapter 5, there are two aspects to replication:

- How data gets replicated around an existing network of links between DCs
- How the Knowledge Consistency Checker generates and maintains the replication links between servers, both intrasite and intersite

We covered the former in Chapter 5, and we'll cover the latter here, leading to an explanation of how to properly design a representation of your organization's network infrastructure within Active Directory.

9.1 Intrasite and Intersite Topologies

Two distinct types of replication links exist with Active Directory sites: intrasite (within sites) and intersite (between sites). An Active Directory service known as the Knowledge Consistency Checker (KCC) is responsible for automatically generating the replication links between intrasite DCs. The KCC will create intersite links automatically for you but only when an administrator has specified that two sites should be connected. Every aspect of the KCC and the links that are created is configurable, so you can manipulate what has been automatically created and what will be automatically created via manipulation of the various options. You can even disable the KCC if you wish and manually create all links.

Note that there is a large distinction between the KCC (the process that runs every 15 minutes and creates the replication topology) and the replication process itself. The KCC is not involved in the regular work of replicating the actual data in any way. Intrasite replication along the links created by the KCC uses a notification process to announce that changes have occurred. So each domain controller is responsible for notifying its replication partners of changes. If no changes occur at all within a 6-hour period, the replication process is kicked off automatically anyway just to make sure. Intersite replication, on the other hand, does not use a notification process. Instead it uses a replication schedule to transfer updates, using compression to reduce the total traffic size.

The KCC and the topologies it generates have been dramatically improved in Windows Server 2003 Active Directory. With Windows 2000 Active Directory, when there were more than 200 sites with domain controllers, it could take the KCC longer than 15 minutes to complete and also drive up CPU utilization. Since the KCC runs every 15 minutes, it could get backlogged or not finish. Typically when faced with this situation, administrators had to disable the KCC and manually create connection objects. With Windows Server 2003, Microsoft has stated that the new limit is closer to 5,000 sites when running a forest at the Windows Server 2003 forest functional level, which is a vast improvement. In fact, the KCC was largely rewritten in Windows Server 2003 and is much more scalable and efficient.

However, we don't think as an Active Directory administrator you should just accept the topologies it creates without examining them in detail. You should investigate and understand what has been done by the KCC. If you then look over the topology and are happy with it, you have actively, rather than passively, accepted what has been done. While letting the KCC do its own thing is fine, every organization is different, and you may have requirements for the site and link design that it is not aware of and cannot build automatically.

Other administrators will want to delve into the internals of Active Directory and turn off the KCC entirely, doing everything by hand. This approach is valid, as long as you know what you're doing, but we prefer to let the KCC do its work, helping it along with a guiding hand every now and then. We cover all these options in the design section later.

9.1.1 The KCC

DCs within sites have links created between them by the KCC. These links use the DC's GUID as the unique identifier. These links exist in Active Directory as connection objects and use only the Directory Service Remote Procedure Call (DS-RPC) transport to replicate with one another. No other replication transport mechanism is available. However, when you need to connect two sites, you manually create a site link via the Active Directory Sites and Services MMC snap-in and specify a replication transport to use. When you do this, the Intersite Topology Generator (ISTG) automatically creates connection objects in Active Directory between domain controllers in the two sites. Within each site, an ISTG is designated to generate the intersite topology for that particular site via the KCC process. There are two replication transports to choose from when creating a site link: standard DS-RPC or Inter-Site Mechanism Simple Mail Transport Protocol (ISM-SMTP). The latter means sending updates via the mail system using certificates and encryption for security.

There are two reasons that the ISTG cannot automatically create links between two sites. First, the ISTG has no idea which sites you will want to connect. Second, the ISTG does not know which replication transport protocol you will want to use.

The KCC runs locally every 15 minutes on each DC. The default time period can be changed, and it can be started

9.2 Designing Sites and Links for Replication

There is only one really important point, which is the overriding factor when designing a replication strategy for your network: how much traffic and over what period will you be replicating across the network? However, replication isn't the only reason for creating sites. Sites also need to exist to group sets of machines together for ease of locating data, finding the nearest DC to authenticate with, or finding the nearest DFS share mount point.

9.2.1 Step 1—Gather Background Data for Your Network

Before you sit down to design your site and WAN topology, you need to obtain the map of your existing network infrastructure. This map should contain all physical locations where your company has computers, along with every link between those locations. The speed and reliability of each link should be noted.

If you have an existing IP infrastructure, write down all the subnets that correspond to the sites you have noted.

9.2.2 Step 2—Design the Sites

From the network diagram, you need to draw your site structure and name each site, using a one-to-one mapping from the network diagram as your starting point. If you have 50 physical WAN locations, you have 50 sites. If only 30 of these will be used for Active Directory, you may not see a need to include the entire set of sites in Active Directory. If you do include the entire set, however, it is much easier to visualize your entire network and add clients or servers to those locations later.

When drawing Active Directory networks, sites normally are represented by ovals.

Remember that a site is a well-connected set of subnets (well-connected tends to mean about 10 Mbps LAN speed). A site does not have to have a server in it; it can be composed entirely of clients. If you have two buildings—or an entire campus—that is connected over 10/100 Mbps links, your entire location is a single site.

This is not a hard and fast rule. By the normal rules, two locations connected over a 2 Mbps link represent two distinct sites. You can, however, group networks together into single sites if you want to. You have to appreciate that there will be more replication than if you had created two sites and a site link, because DCs in both physical locations will maintain the intrasite replication ring topology. If you had created two sites and a site link, only two bridgehead servers would replicate with each other.

We've also successfully used a single site to represent two networks, one with clients and one with servers, separated by a 2 Mbps link. The clients at the end of the 2 Mbps link successfully authenticated quickly and downloaded profiles from a server at the other end of the other link. If we'd used two sites, we would have had to create a site link between them, but the clients still would have had to authenticate across the link anyway.

To summarize, we would suggest that, by default, you create one site per 10 Mbps or higher location, unless you have an overriding reason not to do so.

9.2.3 Step 3—Design the Domain Controller Locations

Placing of DCs is fairly easy, but the number of DCs to use is a different matter entirely.

9.2.3.1 Where to put DCs

Each workstation in a domain exists in a single site that it knows about. When a user tries to log on to the domain at that workstation, the workstation authenticates to a DC from the local site, which it originally locates via a DNS query. If no DC is available in the local site, the workstation finds a remote site, and by a process of negotiation with a DC in that site, either authenticates with that DC or is redirected to a more local DC.

9.3 Examples

Having considered the 10 steps, let's take another brief look at the 3 examples from the previous chapter and see what they will need in terms of sites.

9.3.1 TwoSiteCorp

TwoSiteCorp has two locations split by a 128 Kbps link. This means creation of two sites separated by a single site link, with DCs for domain authentication in each site. The site link cost is not an issue, as only one route exists between the two sites. Here the only issue is scheduling the replication, which depends on the existing traffic levels of the link. Schedule replication during the least busy times for a slow link like this. If replication has to take place all the time, as changes need to be propagated rapidly, it is time to consider increasing the capacity of the link.

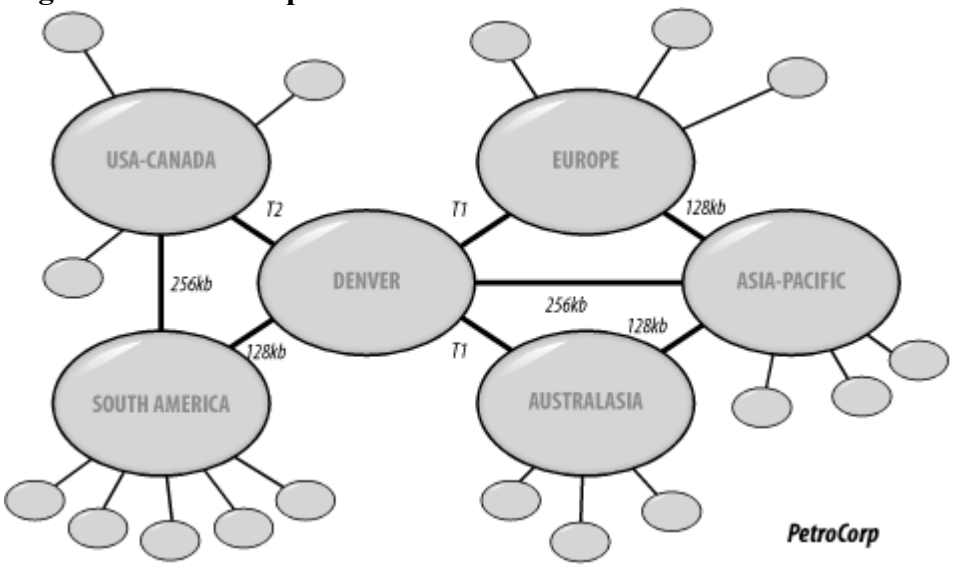
9.3.2 RetailCorp

RetailCorp has a large centralized retail organization with 600 shops connected via 64 Kbps links to a large centralized 10/100 Mbps interconnected headquarters in London. In this situation, you have one site for HQ and 600 sites for the stores. RetailCorp also uses a DC in each store. They then have to create 600 high-cost site links, each with the same cost. RetailCorp decides this is one very good reason to use ADSI (discussed in Part III) and writes a script to automate the creation of the site link objects in the configuration. The only aspect of the site links that is important here is the schedule. Can central HQ cope with all of the servers replicating intersite at the same time? Does the replication have to be staggered? The decision is made that all data has to be replicated during the times that the stores are closed; for stores that do not close, data is replicated during the least busy times. There is no need to worry about site link bridges or site link transitiveness as all links go through the central hub, and no stores need to intercommunicate. The administrators decide to let the KCC pick the bridgehead servers automatically.

9.3.3 PetroCorp

PetroCorp has 94 outlying branch offices. These branch offices are connected via 64 Kbps links to 5 central hub sites. These 5 hubs are connected to the central organization's HQ in Denver via T2, T1, 256 Kbps, and 128 Kbps links. Some of the hubs also are interconnected. To make it easier to understand, look at PetroCorp's network again (Figure 9-8).

Figure 9-8. PetroCorp's network connections



Initially, you need to create 100 sites representing HQ, the hubs, and the branch offices. How many servers do you need per site? From the design we made in Chapter 8, we decided on 9 domains in the forest. Each of those distinct domains must obviously have a server within it that forms part of the single forest. However, although the description doesn't say so, there is very little cross-pollination of clients from one hub needing to log on to servers from another hub. As this is the case, there is no need to put a server for every domain in every hub. If a user from Denver travels to the asiapac.petrocorp.com domain, the user can still log on to petrocorp.com from the Asia-Pacific hub, albeit

9.4 Summary

After this chapter, you should have more of an insight into creating the site and replication infrastructure for your own Active Directory network. Having a basic understanding of the replication process (from Chapter 5) and how the KCC operates should allow you to make much more informed judgments on how much control you want to exert over the KCC in your designs. We feel that it is always better to give free reign to the KCC if possible, while maintaining a firm grip over what it has authority to do. While this can seem contradictory, we hope that our explanations on using site link bridges and restricting transitiveness when appropriate show how this is possible in practice.

The next chapter deals with how to update your designs to reflect your requirements for Group Policy Objects in your organization.

Chapter 10. Designing Organization-Wide Group Policies

This chapter takes an in-depth look at Group Policy Objects (GPOs), focusing on three areas:

- How GPOs work in Active Directory
- How to manage GPOs with the Group Policy Object Editor and Group Policy Management Console
- How to structure your Active Directory effectively using Organizational Units and groups so that you can make the best use of the GPOs required in your organization.

10.1 How GPOs Work

Group policies are very simple to understand, but their uses can be quite complex. Each GPO can consist of two parts: one that applies to a computer (such as a startup script or a change to the system portion of the registry) and one that applies to a user (such as a logoff script or a change to the user portion of the registry). You can use GPOs that contain only computer policies, only user policies, or a mixture of the two.

10.1.1 How GPOs Are Stored in Active Directory

GPOs themselves are stored in two places: Group Policy Configuration (GPC) data is stored in Active Directory, and certain key Group Policy Template (GPT) data is stored as files and directories in the system volume. They are split because while there is definitely a need to store GPOs in Active Directory if the system is to associate them with locations in the tree, you do not want to store all the registry changes, logon scripts, and so on in Active Directory itself. To do so could greatly increase the size of your DIT file. To that end, each GPO consists of the object holding GPC data, which itself is linked to a companion directory in the system volume that may or may not have GPTs stored within. The GPT data is essentially a folder structure that stores Administrative Template-based policies, security settings, applications available for software installation, and script files. GPT data is stored in the System Volume folder of DCs in the *Policies* subfolder.

Third-party developers can extend GPOs by incorporating options that do not reside in the normal GPT location.

The GPO objects themselves are held as instances of the groupPolicyContainer class within a single container in Active Directory at this location:

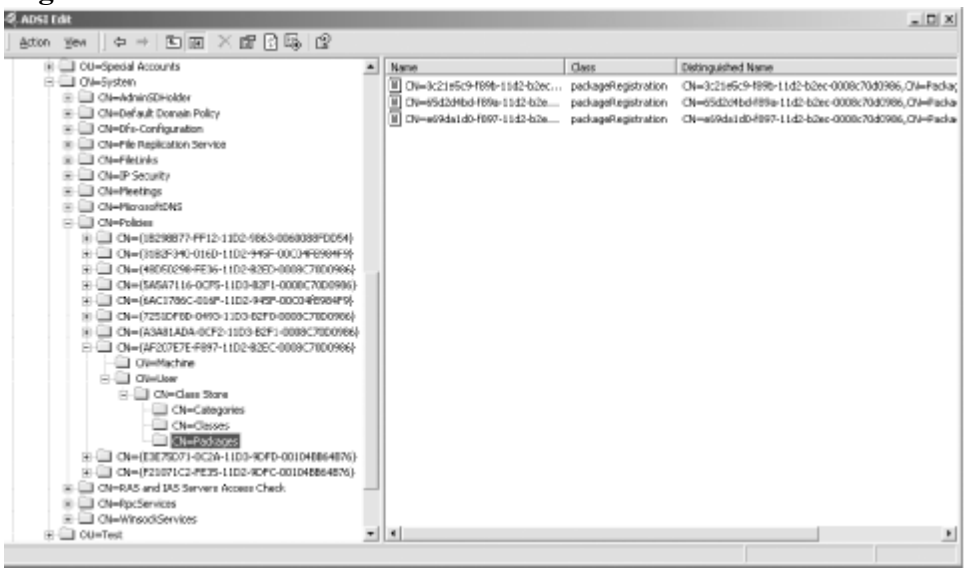
```
CN=Policies,CN=System,dc=mycorp,dc=com
```

Through a process known as linking, the GPOs are associated with the locations in the tree that are to receive the group policy.[1] In other words, one object can be linked to multiple locations in the tree, which explains how one GPO can be applied to many Organizational Units, sites, or domains as required.

[1] The GPC data part of a GPO is an object in Active Directory. This object, like all others, has attributes. One of the attributes of a GPO is a multivalued attribute called gPLink that stores the DN of the containers that the GPO is linked to.

Let's consider the groupPolicyContainer class objects themselves. Take a look at Figure 10-1; we are using one of the Windows Support Tools utilities, ADSI Edit, to show the view of the Policies container and its children.

Figure 10-1. GPOs in the Policies container



10.2 Managing Group Policies

The Microsoft tools available to manage GPOs under Windows 2000 were pretty limited, consisting of the Group Policy Object Editor (formerly Group Policy Editor) and built-in support in the Active Directory Users and Computers and Active Directory Sites and Services snap-ins. While these tools could get the job done, they did not provide any support for viewing the Resultant Set of Policy (RSoP), viewing how GPOs had been applied throughout a domain, or backing up or restoring GPOs. Luckily these tools weren't the only option: third-party vendor Full Armor produced Fazam 2000, which has comprehensive group policy management functionality.

Directly after the release of Windows Server 2003, Microsoft released the Group Policy Management Console (GPMC) as a separate web download. The GPMC is a much-needed addition to Microsoft's GPO management tools and provides nearly every GPO management function that an organization might need, including scripting support.

The other new feature available in the Windows Server 2003 Active Directory administrative tools and in GPMC is support for viewing the RSoP for a given domain, site or Organizational Unit based on certain criteria. RSoP allows administrators to determine what settings will be applied to a user and can aid in troubleshooting GPO problems. RSoP will be described in more detail in the section on debugging group policies.

10.2.1 Using the Group Policy Object Editor

When you add a GPOE snap-in to a console, you can only focus on a particular GPO/LGPO. Each GPO/LGPO that you wish to change has to be loaded in as a separate GPOE snap-in to the MMC; unfortunately, you can't tell the GPOE to show you all policies in the tree, but you can use the GPMC for that.

Managing LGPOs is done using the same GPOE tool that you would use to manage GPOs. If you use the GPOE from a workstation or server in a domain, you can focus the snap-in to look at an LGPO on a local client. If you use the GPOE on a standalone server or a workstation, the GPOE will automatically focus on the LGPO for that machine. No matter how the focus is shifted to look at an LGPO, the GPOE will load only the extensions that are appropriate to the templates in use locally on that client. Domain-specific extensions are not loaded for LGPOs.

GPOs and the PDC FSMO Role Owner

When you are editing GPOs, the GPOE connects to and uses the FSMO PDC role owner. This ensures that multiple copies of the GPOE on different machines are all focused on the same DC. This behavior may be overridden in two cases.

If the PDC is unavailable for whatever reason, an error dialog will be displayed, and the administrator may select an alternate DC to use.

Microsoft is also currently considering a GPOE View menu option and/or a policy to allow the GPOE to inherit from the DC that the Active Directory Users and Computers MMC is focused on. This is likely to be most useful when there is a slow link to the PDC.

If GPOs are edited on multiple DCs, this could lead to inconsistencies because the last person to write to the GPO wins. For this reason, you should use caution when multiple administrators regularly administer policies.

Starting an MMC and adding the GPOE snap-in is not the normal method of accessing GPOs. In fact, there is a whole extended interface available from the Active Directory Sites and Services snap-in, Active Directory Users and Computers (ADUC) tool, or the group Policy Management Console. If you open up the Sites and Services snap-in, you can right-click any site and from the drop-down list select Properties, finally clicking the Group Policy tab on the resulting property page. If you open the ADUC, right-click any domain or Organizational Unit container and follow

10.3 Using GPOs to Help Design the Organizational Unit Structure

In Chapter 8, we described the design of the Active Directory Organizational Unit hierarchy. We also explained that other items have a bearing on that design. You see, there are two key design issues that affect the structure of your Organizational Units: permissions delegation and GPO placement. If you decide that your Active Directory is to be managed centrally rather than in a distributed fashion and that you will employ only a few GPOs that will be implemented mostly domainwide (rather than many GPOs on many Organizational Units), your Organizational Unit structure can be almost any way that you want it to be. It shouldn't make much difference whether you have 400 branches coming off the root or one container with every item inside it. However, if permissions over specific objects do need to be delegated to specific sets of administrators, it will make more sense to structure your domain Organizational Units in a manner that facilitates that administration. This doesn't have to be the case, but it makes it much easier to use Organizational Units.

For example, if we have 1,000 users and 10 managers who each manage 100 users, we could put the 1,000 users in one Organizational Unit and give the 10 admins permission to modify only their 100 users. This is a slow and daft way to run systems administration. It would be better to create 10 Organizational Units and put 100 users in each, giving each administrator permission over his particular Organizational Unit. This makes much more sense, as the administrator can be changed very easily, it is easier to report on access, and so on. Sense and reducing management overhead are the overriding keys here. Either solution is feasible; one is just easier to implement and maintain.

Permissions delegation is covered in more detail in Chapter 11.

The same fundamental facts apply to GPOs. If you are going to need to apply multiple policies to multiple sets of users, it makes more sense and will be easier to manage if you set up multiple Organizational Units. However, this isn't always possible, for example, if the Organizational Unit structure that you have as an ideal conflicts with the one that you will need for permissions delegation, which again conflicts with the one you would like for GPO structuring.

10.3.1 Identifying Areas of Policy

We will assume that within your organization, you will be writing a document that describes your plan for the security features you wish to use in your Active Directory environment and exactly how those features will be implemented. Part of this document will relate to other security features of AD, such as Kerberos, firewalls, permissions, and so on, but here we're concerned with GPOs.

First you need to identify the general policy goals that you wish to achieve with GPOs. There's no need to go into the exact details of each GPO setting and its value at this moment. Instead, you're looking at items like "Deploy financial applications" and "Restrict desktop settings." As you identify each general policy area, you need to note whether it is to apply to all computers or users in a site, to all computers or users in a single domain, or to a subsection of the user and computer accounts. If you aren't sure for some items, put the items in more than one category. You end up with items like "Deploy financial applications to accountants in France" and "Restrict desktop settings in southern Europe."

Once you have the general policy areas constructed, you need to construct an Organizational Unit structure that facilitates implementation of this policy design. At this point, you start placing computers and users in various Organizational Units, deciding if all objects in each container are to receive the policy or whether you will restrict application to the policy via ACLs. There are a number of questions you can ask yourself during this stage. To help with this, a loose set of guidelines follows the example in the next section.

Ultimately the document will need to specify exactly which GPO settings are to be applied, which groups you will set up for ACL permission restrictions, and what the Organizational Unit structure is going to be. It helps to explain justifications for any decisions you make.

To make the guidelines more meaningful, we'll show how you can structure a tree in different ways using a real-world example.

10.4 Debugging Group Policies

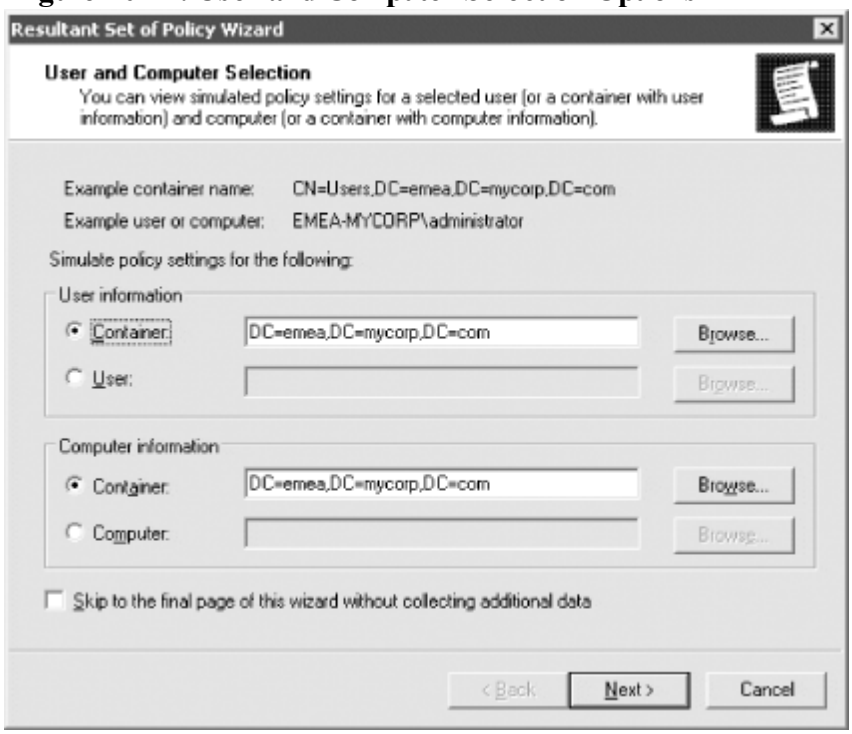
If at any point you need to debug group policies, there are couple of options you can use. The first is new to Windows Server 2003 and is called the Resultant Set of Policy, which some people may be familiar with if you've used tools like Full Armor's Fazam 2000. The Resultant Set of Policy (RSoP) allows you to specify certain user, computer, group, and GPO criteria to determine what will be applied. Another option is to enable some extra logging that can help point out GPO processing problems.

10.4.1 Using the RSoP

The RSoP is a very powerful tool to help identify what GPO settings will be applied to a user or computer. Before RSoP, administrators were left to do their own estimates as to what GPOs took precedence and what settings were actually applied to users and computers. RSoP removes much of the guesswork with an easy-to-use wizard interface.

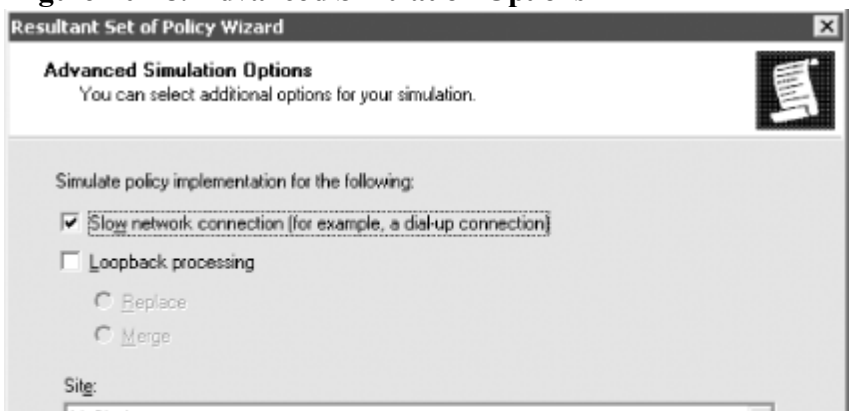
To start the RSoP wizard, open Active Directory Users and Computers and browse to the domain or Organizational Unit that contains the users you want to simulate. Right click on the container and select All Tasks ⟶ Resultant Set Of Policy (Planning). Figure 10-17 shows the initial screen.

Figure 10-17. User and Computer Selection Options



You must first select a specific object DN of a user or computer, an Organizational Unit that contains users or computers, or a domain. After clicking Next, you will come to the Advanced Simulation Options screen where you can select whether to pretend you are over a slow network, whether to use loopback mode, and whether a specific site should be used. Figure 10-18 shows what this screen looks like with the MySite1 site selected.

Figure 10-18. Advanced Simulation Options



10.5 Summary

One of the big selling points of Active Directory has always been group policy and in Windows Server 2003 Active Directory, Microsoft extended the functionality and management of GPOs greatly. In this chapter we expanded on the information presented in Chapter 7, to cover the details of how group policies are stored in Active Directory, how GPOs are processed by clients, the GPO precedence order, the effect of inheritance, and the role ACLs play.

With Windows Server 2003, Microsoft provided several new tools to help manage and troubleshoot GPOs. Perhaps the most important is the Group Policy Management Console (GPMC), which is a one-stop shop for all your GPO needs. With the GPMC you can perform virtually any function you need to do from a single interface, as opposed to using three or four as wa necessary with the Windows 2000 tools. Another benefit of the GPMC is that is installs several COM objects that allow you to script 90% of your GPO management functions. Another long-awaited feature that is available now is the Resultant Set of Policy (RSoP) that allows for modeling and testing of GPOs. With RSoP you can configure several different settings including the container to process, any security groups to include, whether to use a specific site, whether to use loopback mode, whether to use a specific WMI filter, and more. The end result is a GPOE view of the settings that would be applied.

Chapter 11. Active Directory Security: Permissions and Auditing

Permissions can be set in Active Directory in the same way they are set for files. While you may not care that everyone in the tree can read all your users' phone numbers, you may want to store more sensitive information and restrict that access. Reading is not the only problem, of course. You also have create, modify, and delete privileges to worry about, and the last thing you need is a disgruntled or clever employee finding a way to delete all the users in an Organizational Unit. And inheritance increases the complexity in the typical way.

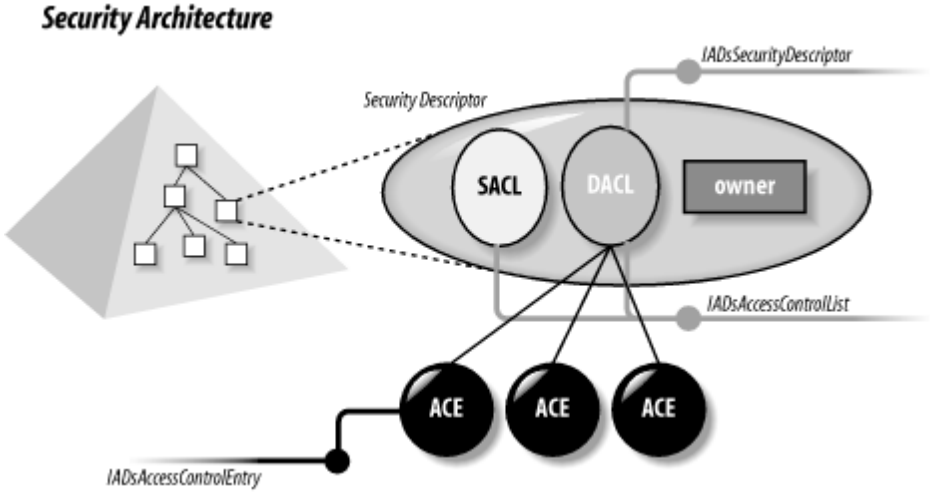
None of this should be new to system managers who already deal with Windows NT Access Control Lists and Access Masks, IntraNetWare's Trustee Lists and Inherited Rights Masks, and Unix's access permissions in file masks. In fact, Microsoft has carried the NT terminology from file permissions forward to Active Directory, so if you already know these terms, you're well ahead. If you are not familiar with them, don't worry. Microsoft has a great tradition of calling a shovel a ground-insertion-earth-management device. Terminology in permissions can seem confusing at first, so we'll go through it all in detail.

Managing the permissions in Active Directory doesn't have to be a headache. You can design sensible permissions schemes using guidelines on inheritance and complexity that will allow you to have a much easier time as a systems administrator. The GUI that Microsoft provides is fairly good for simple tasks but more cumbersome for complex multiple permissions. In Windows Server 2003, the GUI has been enhanced to provide an "effective permissions" option that lets you determine the effective permissions a user group has on the container or object. Also, Active Directory permissions are supported by ADSI, which opens up a whole raft of opportunities for you to use scripts to track problems and manipulate access simply and effectively. Finally, the DSACLS utility allows administrators to manage permissions from a command line if you prefer an alternative to the GUI

Yet permissions are only half the story. If you allow a user to modify details of every user in a specific branch below a certain Organizational Unit, you can monitor the creations, deletions, and changes to objects and properties within that branch using auditing entries. In fact, you can monitor any aspect of modification to Active Directory using auditing. The system keeps track of logging the auditing events and you can then periodically check them or use a script or third-party tool to alert you quickly to any problems.

Figure 11-1 shows the basics. Each object stores a value called a Security Descriptor, or SD, that holds all the information describing the security for that object. Included with the information are two important collections called Access Control Lists, or ACLs, which hold the relevant permissions. The first ACL, called the System-Audit ACL or SACL, defines the permission events that will trigger both success and failure audit messages. The second, called the Discretionary ACL or DACL, defines the permissions that users have to the object, its properties, and its children. Each of the two ACLs holds a collection of Access Control Entries, or ACEs, that correspond to individual audit or permission entries.

Figure 11-1. Active Directory security architecture



11.1 Using the GUI to Examine Permissions

To access the default permissions for any object, select the Active Directory Users and Computers MMC and right-click on it. Choose Properties from the drop-down menu and select the Security tab of the properties window that is displayed.

To make the Security tab visible, you need to right-click in the display pane of the Active Directory Users and Computers MMC, choosing View Advanced Features from the pop-up menu. If you reopen the properties window of the object to which you wish to assign permissions, you should see a Security tab.

The window in Figure 11-2 is your first point of contact for permissions. The top area contains a complete list of all groups and users who have permissions to the object whose properties we are looking at. The Permissions section below this list displays which general permissions are allowed and denied for the highlighted user or group. The general permissions listed are only those deemed to be the most likely to be set on a regular basis. Each general permission is only an umbrella term representing a complex set of actual implemented permissions hidden underneath the item. Consequently, the general permission called Read translates to specific permissions like Read All Properties and List Contents, as we will show later. Below the Permissions section are three important parts of this window:

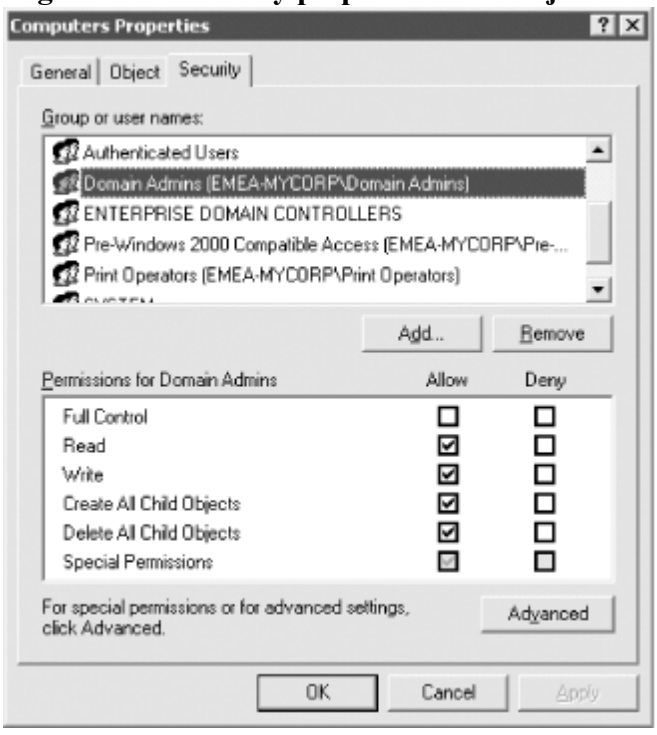
Advanced button

The Advanced button allows you to delve further into the object, so that permissions can be set using a more fine-grained approach.

Text display area

The second part of this area of the window is used to display a message, such as that shown in Figure 11-2. The text shows that the permissions for the current object are more complex than can be displayed here. Consequently, we would have to press the Advanced button to see them.

Figure 11-2. Security properties of an object



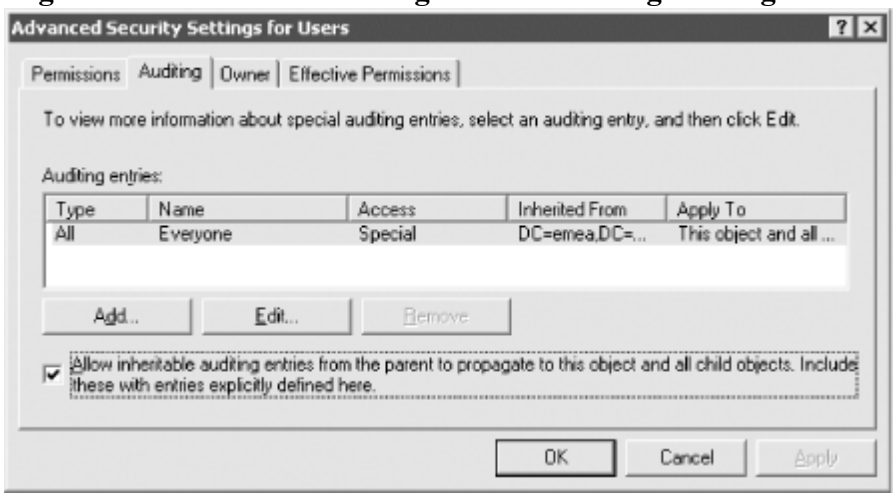
Inheritance checkbox (Windows 2000 only; not shown in Figure 11-2)

The "Allow inheritable permissions from parent to propagate to this object" checkbox allows you to orphan (my term) this object from the tree. When you clear the checkbox on the security properties or Access Control Settings windows mentioned later, the system pops up a Yes/No/Cancel dialog box that asks if you want to convert your inherited entries to normal entries. If you click Cancel, the operation aborts. Clicking No removes all inherited entries

11.2 Using the GUI to Examine Auditing

Examining auditing entries is almost identical to viewing permissions entries. If you go back to the screen shown in Figure 11-3 and click on the Auditing tab, a screen similar to that in Figure 11-11 is displayed.

Figure 11-11. Advanced Settings window showing auditing entries



This window shows the list of Auditing Entries (AEs) that have been defined on the object. This object has one AE, and it's not very helpful viewing it from here since the detail is too limited. So just as you would do with permissions, you can click the Edit button (or View/Edit with Windows 2000), drill down, view the individual AE itself.

Figure 11-12 shows the successful and failed items that are being awaited. The items are grayed out because this entry is inherited from further up the tree, i.e., it is not defined directly on this object but instead further up the hierarchy.

Figure 11-12. Auditing entry for an object

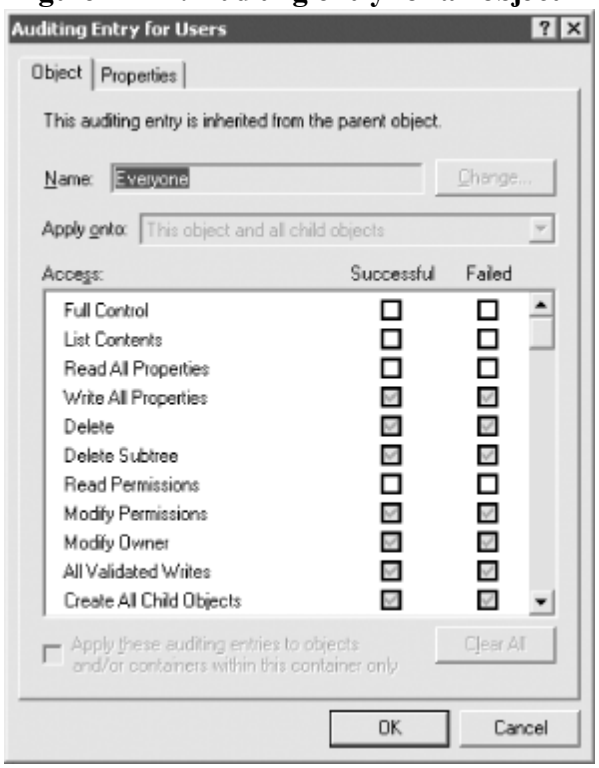


Figure 11-13 shows an example AE window for successful and failed auditing of properties. Here you are auditing only property writes.

Figure 11-13. Auditing entry for an object's properties

11.3 Designing Permission Schemes

Having worked through many designs for different domain structures, we have come up with a series of rules or guidelines you can follow to structure the design process effectively. The idea is that if you design your permissions schemes using these rules, you will be more likely to create a design with global scope and minimum effort.

11.3.1 The Five Golden Rules of Permissions Design

This list is not exhaustive. We are sure you will be able to think of others beyond these. If, however, these rules spark your creative juices and help you design more effectively, they will have done their job.

The rules are:

1. Whenever possible, assign object permissions to groups of users rather than individual users.
2. Design group permissions so that you have a minimum of duplication.
3. Manage permissions globally from the ACL window.
4. Allow inheritance: do not orphan sections of the tree.
5. Keep a log of every unusual change that you have made to the tree, especially when you have orphaned sections of it or applied special rights to certain users.

Let's look at these rules in more detail.

11.3.1.1 Rule 1—Apply permissions to groups whenever possible

By default, you should use groups to manage your user permissions. At its simplest, this rule makes sense whenever you have more than one user for whom you wish to set certain permissions.

Global Group and Local Group Permissions Under Windows NT 4.0

Under Windows NT 4.0, Microsoft's preferred method of applying file and directory permissions was to create two sets of groups: Local Groups, which had permissions, and Domain Global Groups, which contained users.

The Local Group would exist on the server that had the resource, and the relevant permissions were assigned to that. Local groups were allowed to contain both users and groups. Domain Users were then placed in Domain Global Groups, which themselves were placed in the Local Groups on each server. Domain Global Groups were allowed to contain only users and not other groups. This may sound complicated, but it worked well in practice. A good way of demonstrating this is through an example.

Consider an NT 4.0 domain called Mycorp containing a Global Domain Group called Marketing. This group has four members. Within Mycorp are two servers, called Server1 and Server2, each of which has published a share. Each server also has a Local Group SH_USERS, which contains the Global Group Marketing as a member. Each SH_USERS group has read access to the relevant share on the

11.4 Designing Auditing Schemes

Designing auditing schemes, in contrast to permissions, is a relatively easy process. Imagine the circumstances in which you may need to check what is happening in Active Directory, and then set things up accordingly.

You must remember that every Active Directory event that is audited causes the system to incur extra processing. Having auditing turned on all the time at the root for every modification by anyone is a great way to get all DCs to really slow down if a lot of Active Directory access occurs on those DCs.

That point bears repeating. Auditing changes to anywhere in the domain Naming Context (NC) will propagate domainwide and cause logging to the security event log on every DC that services the Domain NC. Auditing changes to the Configuration NC or Schema NC will cause all DCs in a forest to begin auditing to their security event logs. You must have tools in place to retrieve logs from multiple DCs if you wish to see every security event that occurs. After all, if you have 100 DCs and are logging Configuration NC changes, then because changes can occur on any DC, you need to amalgamate 100 security event logs to gather a complete picture.[1]

[1] Applications for consolidation of event logs are SeNTry by Mission Critical, Event Admin by Aelita, and AppManager by NetIQ. Also, note that Microsoft's WMI technology has excellent event logging, reporting, and notification capabilities if you wish to script such items yourself.

Here are a few examples where designing auditing schemes could come in handy:

- Someone complains that user details are being set to silly values by someone else as a joke.
- You notice that new objects you weren't expecting have been created or deleted in a container.
- The Active Directory hierarchy has changed and you weren't informed.
- You suspect a security problem.

In all these scenarios, you will need to set auditing options on a container or a leaf object. These auditing entries do not have to exist all the time, so you could write them up and then code them into a script that you run at the first sign of trouble. That way, the system is immediately updated and ready to monitor the situation. This can happen only if you are prepared.

You need to analyze the scenarios that you envisage cropping up and then translate them into exact sets of auditing entry specifications. After you have written up each scenario and an emergency occurs, you will be able to follow the exact instructions that you previously laid down and set up a proper rapid response, which is what auditing is all about.

Step one in a real emergency may be to turn all auditing on at the root to make sure that you capture everything to the security log. Step two may be to turn on auditing for the specific items that you need to audit, so that with step three you can finally remove the Audit-All at the root that normally would cause a severe slowdown. That way, you slow Active Directory briefly while setting up the auditing you actually require, but you don't lose any audit entries during that time. The point is that having a properly prepared set of scripts will save you trouble in the long run as you can quickly use your "Audit all object creations and deletions below a container" or "Audit this object only for any

11.5 Real-World Examples

It now seems appropriate to put what we have laid out earlier into practice. We will use a series of tasks that could crop up during your everyday work as an administrator. The solutions we propose probably are not the only solutions to the problems. That is often the case with Active Directory; there are many ways of solving the same problem.

11.5.1 Hiding Specific Personal Details for All Users in an Organizational Unit from a Group

In this example, an Organizational Unit called Hardware Support Staff contains the user accounts of an in-house team of people whose job is to repair and fix broken computers within your organization. Whenever a user has a fault, he rings the central faults hotline to request a support engineer. An engineer is assigned the job, which is added to the end of her existing list of faults to deal with. The user is informed about which engineer will be calling and approximately when she will arrive. As with all jobs of this sort, some take longer than others, and users have discovered how to find the mobile or pager number of the engineer they have been assigned and have taken to calling her to get an arrival time update rather than ringing the central desk. Management has decided that they want to restrict who can ring the pager or mobile, but they do not want to stop giving out the engineer's name as they feel it is part of the friendly service. Consequently, they have asked you to find a way of hiding the pager and mobile numbers in Active Directory from all users who should not have access.

The solution that we will use is to remove the default ability of every user to see the property of the Engineer objects by default. We can do this either from the parent OU or manually for each engineer. This ensures that only users or groups that we allow to see the properties will do so. Since this is a simple problem with a simple solution, it is easier to use the GUI than to write a script.

We start by creating a group for users who are allowed to see these properties, calling it Support Phone or something similar. Now we have to make the decision: do we select the parent Organizational Unit itself and assign permissions to hide the property for objects within the container and down the tree, or do we manually apply permissions to every support engineer's account? The latter is likely to take much longer with any reasonable number of support staff, and it comes with the added problem that we will have to do the same tasks every time a new support staff member joins the team. In this instance, we will choose the former; however, it should be noted that this will hide all the mobile and pager numbers of all users under the Hardware Support Staff Organizational Unit, even if some of them are not engineers. This is covered in the next example.

We open the ACS window for the Hardware Support Staff Organizational Unit and click Add. We then locate the Support Phone group and click OK. This opens the PE window for Support Phone relating to the Organizational Unit. Now we click the Properties tab, specify to apply to this object and all subobjects, and then click Allow for both the properties Read Phone-Pager-Primary and Read Phone-Mobile- Primary. These two items may already be allowed by default. If we now click OK, the permissions are applied down the tree, so that everyone in the Support Phone group can now read the mobile and pager properties of all user objects below that Organizational Unit.

We need to restrict the rest of the tree from viewing these two properties. From the ACS window for the Organizational Unit, we add a group in the same manner as before, this time specifying Everyone as the group. We select the Properties tab, find the Read Phone-Pager-Primary and Read Phone-Mobile-Primary properties, and remove the check marks which occur in the two allow fields. We click OK, and all members of the group Everyone have no rights to the two properties below this Organizational Unit. This differs from specifically denying all members access.

If we had denied the Everyone group from reading the two properties as our first step, then when we opened up the PE window for the Support Phone group, it would not have had the existing check marks inside it for the two fields. This is because Active Directory would already have realized that the members of Support Phone were obviously members of Everyone, the group containing every user on the system, and consequently would have removed the two settings.

11.6 Summary

Security is always important, and when access to your organization's network is concerned, it's paramount. We hope this chapter has given you an understanding of how permission to access can be allowed or denied to entire domains or individual properties of a single object. Auditing is also part of security, and having mechanisms already designed—so that they can be constantly working or dropped in when required—is the best way to keep track of such a system.

Assigning permission and auditing entries to an object appears to be a simple subject on the surface. However, once you start delving into the art of setting permissions and auditing entries, it quickly becomes obvious how much there is to consider. Global design is the necessary first step.

While expanding your tree later by adding extra containers is rarely a problem, in a large tree it makes sense to have some overall guidelines or rules that allow you to impose a sense of structure on the whole process of design and redesign. Ideally, the golden rules and tables that we created should allow you to plan and implement sensible permissions schemes, which was the goal of the chapter.

Chapter 12. Designing and Implementing Schema Extensions

For Active Directory to hold any object, such as a user, it needs to know what the attributes and characteristics of that object are. In other words, it needs a blueprint for that object. The Active Directory schema is the blueprint for all classes, attributes, and syntaxes that potentially can be stored in Active Directory.

The default schema definition is defined in the *%systemroot%\ntds\schema.ini* file that also contains the initial structure for the *ntds.dit* (Active Directory database). This file contains plain ASCII file and can be viewed using Notepad or any text editor.

The following considerations should be kept in mind when you contemplate extending your schema:

- Microsoft designed Active Directory to hold the most common objects and attributes you would require. Because they could never anticipate every class of object or every specific attribute (languages spoken, professional qualifications) that a company would need, Active Directory was designed to be extensible. After all, if these objects and properties are going to be in everyday use, the design shouldn't be taken lightly. Administrators need to be aware of the importance of the schema and how to extend it. Extending the schema is a useful and simple solution to a variety of problems. Not being aware of the potential means that you will have a hard time identifying it as a solution to problems you might encounter.
- Designing schema extensions is very important, in part because any new class or attribute that you create in the schema is a permanent addition. While unused objects can be disabled if you no longer require them, they cannot be removed. In Windows 2003 Active Directory, you can redefine schema extensions, but you cannot totally remove them.
- While it is easy to extend Active Directory, it's surprising how many companies are reluctant to implement schema extensions due to concerns over the impact to Active Directory. One of the biggest impediments in Windows 2000 was that anytime the partial attribute set was extended (i.e., an attribute added to the Global Catalog) a full resync had to be done for all Global Catalog servers. Fortunately, Microsoft resolved this in Windows 2003, and a full resync is no longer performed.

This chapter takes you through the process of extending the schema, from the initial design of the changes through the implementation, and discusses how to avoid the pitfalls that can crop up. We then talk about analyzing the choices available and seeing if you can obtain the required design result some other way, because schema changes are not to be undertaken lightly. We obviously cover how to implement schema changes from first principles, but before that we identify the steps in designing or modifying a class or attribute. Finally, we cover some pitfalls to be aware of when administering the schema.

We don't spend much time introducing a large number of specific examples. This is mainly because there's no way we can conceive of every sort of class that you will require. Consequently, for examples we use only one new generic class as well as a few attribute extensions to the default user object. When giving examples of modifying a class, we extend the user object class.

Let's look at how you would design the changes you may wish to make in an enterprise environment.

12.1 Nominating Responsible People in Your Organization

If you don't already have a central person or group of people responsible for the OID namespace for your organization, you need to form such a group. This OID Managers group is responsible for obtaining an OID namespace, designing a structure for the namespace that makes sense to your organization, managing that namespace by maintaining a diagram of the structure and a list of the allocated OIDs, and issuing appropriate OIDs for new classes from that structure as required. Whenever a new class of attribute or object is to be created in your organization's forest, the OID Managers provide a unique OID for that new class, which is then logged by the OID Managers with a set of details about the reason for the request and the type of class that it is to be used for. All these details need to be defined by the OID Managers group.

The Schema Managers, by comparison, are responsible for designing and creating proper classes in the schema for a forest. They are responsible for actually making changes to the schema via requests from within the organization, for ensuring that redundant objects doing the same thing are not created, that inheritance is used to best effect, that the appropriate objects are indexed, and that the GC contains the right attributes.

The Schema Managers need to decide on the membership of the Schema Admins universal group that resides in the Forest Root Domain of a particular forest. One possibility is that the Schema Managers wish to keep a set of user accounts as members of Schema Admins by default all the time. Instead, they may decide to remove every member of the Schema Admins group so that no unintentional changes can be made to the schema. In this case, the Schema Managers need to be given permissions to add and remove members of the Schema Admins group to enable any of the Schema Managers to add themselves to the Schema Admins group whenever changes are to be made to the schema.

If you are designing code that will modify some other organization's schema, the documentation accompanying that code should make it explicitly clear exactly what classes are being created and why. The documentation also should explain that the code needs to be run with the privilege of a member of the Schema Admins group, since some organizations may have an Active Directory in which the Schema Admins group is empty most of the time, as mentioned earlier.

Note that the membership of OID Managers does not necessarily coincide with that of Schema Managers, although it is a possibility.

12.2 Thinking of Changing the Schema

Before you start thinking of changing the schema, you need to consider not just the namespace, but also the data your Active Directory will hold. After all, if you know your data, you can decide what changes you want to make and whom those changes might impact.

12.2.1 Designing the Data

No matter how you migrated to Active Directory, at some point you'll need to determine exactly what data you will add or migrate for the objects you create. Will you use the physicalDeliveryOfficeName attribute of the user object? What about the telephonePager attribute? Do you want to merge the internal staff office location list and telephone database during the migration? What if you really need also to know what languages each of your staff speaks or qualifications they hold? What about their shoe size, their shirt size, number of children, and whether they like animals? The point is that some of these already exist in the Active Directory schema and some don't. At some point you need to design the actual data that you want to include.

Let's consider MyUnixCorp, a large fictional organization that for many years has run perfectly well on a large mainframe system. The system is unusual in that the login process has been completely replaced in-house with a two-tier password system. A file called *additional-passwd* maintains a list of usernames and their second Unix password in an encrypted format. Your design for the migration for MyUnixCorp's system has to take account of the extra login check. In this scenario, either MyUnixCorp accepts that the new Active Directory Kerberos security mechanism is secure enough for its site, or it has to add entries to the schema for the second password attribute and write a new Active Directory logon interface that incorporates both checks.

This example serves to outline that the data that is to be stored in Active Directory has a bearing on the schema structure and consequently has to be incorporated into the design phase.

12.2.2 To Change or Not to Change

When you identify a deficiency in the schema for your own Active Directory, you have to look hard into whether modifying the schema is the correct way forward. Finding that the schema lacks a complete series of objects along with multiple attributes is a far cry from identifying that the Person-who-needs-to-refill-the-printer-with-toner attribute of the printer object is missing from the schema. There's no rule, either, that says that once you wish to create three extra attributes on an existing object, you should modify the schema. It all comes down to choice.

There is one useful guideline: you should identify all the data you wish to hold in Active Directory prior to considering your design. If you consider how to implement each change in Active Directory one at a time, you may simply lose sight of the overall picture.

To help you make that choice, you should ask yourself whether there are any other objects or attributes that you could use to solve your problem.

Let's say you were looking for an attribute of a user object that would hold a staff identification number for your users. You need to ask whether there is an existing attribute of the user object that could hold the staff ID number and that you are not going to use. This saves you from modifying the schema if you don't have to. Take Leicester University as an example. We had a large user base that we were going to register, and we needed to hold a special ID number for our students. In Great Britain, every university student has a so-called University and Colleges Administration System number, more commonly known as the UCAS number, a unique alphanumeric string that UCAS assigns independent of a student's particular university affiliation. Students receive their UCAS numbers when they first begin looking into universities. The numbers identify students to their prospective universities, stay with students throughout their undergraduate careers, and are good identifiers for checking the validity of students' details. By default, there is no schema attribute called UCAS-Number, so we had two choices. We could find an appropriately named attribute that we were not going to use and make use of that, or we could modify the schema.

12.3 Creating Schema Extensions

There are three ways to modify the schema: through the Schema Manager MMC, using LDIF files, or programmatically using ADSI. We will not cover the use of the Schema Manager MMC very heavily here since it is fairly straightforward to use, although we will cover its use in managing the Schema FSMO role. Typically you should not use the Schema Manager MMC to extend the schema and instead use LDIF files or ADSI. Most vendors provide LDIF files, which contain the schema extensions that you can run at your leisure. We cover extending the schema with ADSI in Chapter 24.

12.3.1 Running the Schema Manager MMC for the First Time

The Schema Manager MMC is not available from the Administrative Tools menu like the other Active Directory snap-ins. To use it, you need to first register the Dynamic Link Library (DLL) file for the MMC snap-in by typing the following command at the command prompt:

```
regsvr32.exe schmmgmt.dll
```

You can then start the Schema Manager console by creating a custom MMC and adding the Active Directory Schema snap-in to it. To create a console, go to the Run menu from the Start button, type mmc.exe, and click OK. Then in the empty MMC, choose the Console menu and select Add/Remove Snap-in. From here, you can click the Add button and select Active Directory Schema as the item. If you then click the Add button, followed by Close, and then the OK button, that will give you an MMC hosting the Schema Manager snap-in for you to use and later save as required.

Allowing the Schema to be modified on Windows 2000

Under Windows 2000, there was a safeguard you had to bypass for the Schema FSMO to allow you to modify the schema. With Windows 2003 Active Directory, this is no longer required. First, the user who is to make the changes has to be a member of the Schema Admins group, which exists in the forest root domain. Second, you need to make a change to the registry on the DC that you wish to make the changes on.

The fastest and probably best solution is to use the checkbox from the Schema Master MMC, shown later in the chapter.

Alternatively, on the DC itself, open up the registry using *regedit32.exe* or *regedit.exe* and locate the following key:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters

Now, create a new REG_DWORD value called "Schema Update Allowed" (no quotes) and set the value to 1. That's all you need to do. You now can edit the Schema on that DC.

Another alternative method for making the change is to copy the following three lines to a text file with a REG extension and open it (i.e., execute it) on the DC where you wish to enable schema updates. This will automatically modify the registry for you without the need to open the registry by hand:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters]
"Schema Update Allowed"=dword:00000001
```

Once you've modified the registry on a particular DC and placed the user account that is to make the changes into the Schema Admins group, any changes you make to the schema on that DC will be accepted. If you wish changes to be accepted on any DC, you need to modify the registry correspondingly on every DC.

12.4 Wreaking Havoc with Your Schema

There are a number of ways to cause problems in your Active Directory schema. We include a few examples here so that you can be fully aware of the problems.

Let's start by considering the main base classes of attributeSchema, classSchema, and top. Imagine we decide to add a new mandatory attribute to top. As all classes derive from top, the mandatory attribute requirement is suddenly added to every class and attribute throughout the schema in one go. Since none of the existing classes and attributes have this value, they all suddenly become marked as invalid. They still exist and can be used, but they cannot be modified at all. New timestamps cannot be added, USNs cannot be changed, replication stops, and effectively your Active Directory grinds to a halt. The reason that the objects cannot be modified is that Active Directory does a special check when existing instances of objects are modified to make sure that all mandatory attributes have been set. If they have not all been set, which they won't have been in this case, Active Directory will not allow any attribute changes from now on. The only solution is to remove the new mandatory attribute or set a value for the attribute on every single object in every NC in the entire forest.

There are also concurrency problems. Having a Schema FSMO is perfectly fine, but that doesn't necessarily stop members of Schema Admins from attempting to run two schema-modifying applications at the same time. Every time an application or piece of code attempts to write to the schema, it automatically writes a special system attribute at the same time. Two system-attribute writes anywhere in Active Directory cannot occur simultaneously, so one will fail if this is the case. In the scenario of simultaneous applications executing, the changes to the schema may all be handled sequentially and the requests from both applications may be interleaved, but the two applications at some point may attempt to write together. At that point, one of them will fail. If the failed application is rerun, it must be coded to detect the existence of each object (i.e., the previous creation succeeded) prior to creating the object, or else the object-creation process will continually fail.

You can also make instances of objects invalid quite easily. For example, let's say that we define that new class we mentioned earlier called Finance-User, and create an instance of it called cn=SimonWilliams. If we then remove Languages-Spoken from Finance-User's mandatory attributes, the SimonWilliams user becomes invalid because the SimonWilliams instance has an attribute that is not now allowed in the schema definition for Finance-User. Again, it is up to the person or code that makes the Languages-Spoken attribute defunct to go through Active Directory and find all instances of Finance-User and modify them to remove the value in this now-defunct attribute. If this isn't done, any instances of Finance-User with the Languages-Spoken attribute defined (all, in this case, as it was mandatory) remain invalid.

You cannot cause invalid instances by modifying existing attributeSchema objects, as all the key attributes are defined in system attributes. However, you can cause havoc with existing classSchema objects. Ways of doing this are:

- Removing classes as possible superiors; this can leave instances under invalid parent containers.
- Adding classes to the list of auxiliary classes; this can change what attributes are now considered mandatory.
- Removing classes from the list of auxiliary classes; this can change what attributes are now considered mandatory and optional and can thus leave instances with now nonexistent attributes.
- Directly removing mandatory or optional attributes; this can leave instances with now nonexistent attributes.

If the DC holding the Schema FSMO role unexpectedly disappears, you can force another server to assume the role. But if the original DC ever comes back, you have two Schema FSMOs, and you will need to rectify that by making sure only one server has the role. However, if the original server had some updates applied prior to its crash, and you allow updates to be made on the new Schema Master, the updates from the old DC will eventually propagate around the network. Your problems to be aware of in this scenario are twofold:

12.5 Summary

Carefully designing the changes that you make to the Active Directory schema cannot be stressed highly enough for large corporations or application developers. Selecting a team of Schema Managers and OID Managers and creating documentation to accompany and justify changes will smooth that process. Whether you are a small company or a large multinational, creating sensible structures should mean that you rarely make mistakes and almost never have to make objects defunct.

Hopefully we have shown you not only the perils and pitfalls of modifying the schema but also why the schema is necessary and how it underpins the entire Active Directory. While you should be cautious when modifying Active Directory, a sensible administrator should have as little to fear from the Active Directory schema as he does from the Windows Registry.

Chapter 13. Backup, Recovery, and Maintenance

A very important though often overlooked aspect of maintaining Active Directory is having a solid disaster recovery plan in place. While the reported incidents of corruption of Active Directory have been minimal, it has happened and is something you should be prepared for regardless of how unlikely it is to occur. Restoring accidentally deleted objects is much more likely to happen than complete corruption, and thus you should be equally prepared. Do you have a plan in place for what to do if a domain controller that has a FSMO role suddenly goes offline, and you are unable to bring it back? All the scenarios we've just described typically happen under times of duress. That is, clients are complaining or an application is no longer working correctly and people aren't happy. It is during times like this that you don't want to have to scramble to find a solution. Having well-documented procedures to handle these issues is critical.

In this chapter, we will look at how to prepare for failures by backing up Active Directory. We will then describe how you can recover all or portions of your Active Directory from backup. We will then cover how to recover from FSMO failures. Finally, we will look at other preventive maintenance operations you can do to ensure the health of Active Directory.

13.1 Backing Up Active Directory

Backing up Active Directory is a straightforward operation. It can be done using the NT Backup utility provided with the Windows operating system or with a third-party backup package such as Veritas NetBackup. Fortunately, you can backup Active Directory while it is online, so you do not have to worry about taking outages just to perform backups like you do with other systems, such as Exchange 2000.

To back up Active Directory, you have to back up the System State of one or more domain controllers within each domain in the forest. If you want to be able to restore any domain controller in the forest, you'll need to back up every domain controller. On a domain controller, the System State contains the following:

Active Directory

This includes the files in the NTDS folder that contains the Active Directory database (*ntds.dit*), the checkpoint file (*edb.chk*), transaction log files (*edb*.log*), and reserved transaction logs (*res1.log* and *res2.log*).

Boot Files

The files necessary for the machine to boot up.

COM+ Class Registration Database

The database for registered COM components.

Registry

The contents of the registry.

SYSVOL

This includes the files contained in the NETLOGON share, which typically contain user logon and logoff scripts and system startup and shutdown scripts. It also includes the file-based portion of GPOs, which are stored in SYSVOL.

Certificate Services

This applies only to DCs that are running Certificate Services.

While most backup packages allow you to perform incremental backups, with Active Directory you can only perform full backups of the system state.

The user that performs the backup must be a member of the Backup Operators group or have Domain Admins equivalent privileges.

Due to the way Active Directory handles deleted objects, your backups are only good for a certain period of time. When objects are deleted in Active Directory, initially they are not removed completely. A copy of the object still resides in Active Directory for the duration of the tombstone lifetime. The tombstone lifetime value dictates how long Active Directory keeps deleted objects before completely removing them. The tombstone lifetime is configurable and is defined in the tombStoneLifetime attribute on the following object:

cn=Directory Services, cn=WindowsNT, cn=Services, cn=Configuration, <ForestDN>

The default value for tombStoneLifetime is 60 days. That means deleted objects are purged from Active Directory 2 months after they are initially deleted. As far as backups go, you should not restore a backup that is older than the tombstone lifetime because deleted objects will be reintroduced. If for whatever reason you are not able to get successful backups at least every 60 days, consider increasing the value of tombStoneLifetime.

Another issue to be mindful of in regard to how long you keep copies of your backup has to do with passwords. Computer accounts change their passwords every 30 days. They keep their previous passwords and attempt to use them if their current passwords do not work. So if you restore computer objects from a backup that is older than 60 days, those computers will more than likely not be able to participate in the domain and will have to be reset. Trust relationships can also be affected. Like computer accounts, the current and previous passwords are stored with the

13.2 Restoring a Domain Controller

One of the benefits of Active Directory is built-in redundancy. When you lose a single domain controller, the impact can be insignificant. With many services, such as DHCP, the architecture dictates a dependency on a specific server. When that server becomes unavailable, clients are impacted. Over the years, failover or redundancy has been built into most of these services, including DHCP. With Active Directory, the architecture is built around redundancy. Clients are not dependent on a single DC; they can failover to another DC seamlessly if a failure occurs.

When a failure does occur, you should ask yourself several questions to assess the impact:
Is the domain controller the only one for the domain?

This is the worst-case scenario. The redundancy in Active Directory applies only if you have more than one domain controller in a domain. If there is only one, you have a single point of failure. You could irrevocably lose the domain unless you can get that domain controller back online or restore it from backup.
Does the domain controller have a FSMO role?

The five FSMO roles outlined in Chapter 2 play an important part in Active Directory. FSMO roles are not redundant, so if a FSMO role owner becomes unavailable, you'll need to seize the FSMO role on another domain controller. Check out the FSMO recovery section later in this chapter for more information.
Is the domain controller a Global Catalog server?

The Global Catalog is a function that any domain controller can perform if enabled. But if you have only one Global Catalog server in a site and it becomes unavailable, it can impact user's ability to login. As long as clients can access a Global Catalog, even if it isn't in the most optimal location, they will be able to login. If a site without a Global Catalog for some reason loses connectivity with the rest of the network, it would impact user's ability to login. With Windows Server 2003, you can enable universal group caching on a per-site basis to limit this potential issue.
Is the domain controller necessary from a capacity perspective?

If your domain controllers are running near capacity and one fails, it could overwhelm the remaining servers. At this point, clients could start to experience login failures or extreme slowness when authenticating.
Are any other services, such as Exchange 2000, relying on that specific domain controller?

Early versions of Exchange 2000 did not handle domain controller failures well. In fact, once an Exchange 2000 server targeted a specific domain controller, you would have to manually force it to use another one if that domain controller became unavailable. During the outage period, mail delivery could be impacted along with client lookups. Exchange is just one example, but it illustrates that you have to be careful of this when introducing Active Directory-enabled services into your environment.

These questions can help you assess the urgency of restoring the domain controller. If you answered "no" to all of the questions, the domain controller can stay down for a short period without significant impact.

When you've identified that you need to restore a domain controller, there are two options to choose from: restoring from replication or restoring from a backup.

13.2.1 Restore from Replication

One option for restoring a domain controller is to bring up a freshly installed or repaired machine and promote it into Active Directory. You would use this option if you had a single domain controller failure due to hardware and did not have a recent backup of the machine. This method allows you to replace the server in AD by promoting a newly installed machine and allowing replication to copy all of the data to the DC. Here are the steps to perform this type of restore:

1. Rebuild OS. Reinstall the operating system and any other applications you support on your domain controllers.
2.

13.3 Restoring Active Directory

No one ever wants to be in a position where you have to restore Active Directory, but nevertheless you should prepare for it. Restoring Active Directory comes in a few different flavors, which we'll cover now.

13.3.1 Nonauthoritative Restore

A nonauthoritative restore is a restore where you simply bring a domain controller back to a known good state using a backup. You then let replication resync the contents of the latest changes in Active Directory since the backup. The restore from backup method we described earlier to handle DC failures is an example of a nonauthoritative restore. The only difference between that scenario and the one we'll describe here is that previously we assumed that the failed server you rebuilt or replaced was not a domain controller yet. There may be some circumstances when you want to perform a similar restore, but the server is a domain controller. One example might be if some changes were made on a particular domain controller that you wanted to take back. If you were able to disconnect the domain controller from the network in time before it replicated, you could perform a nonauthoritative restore to get it back to a known state before the changes were made. This would effectively nullify the changes as long as they didn't replicate to another server.

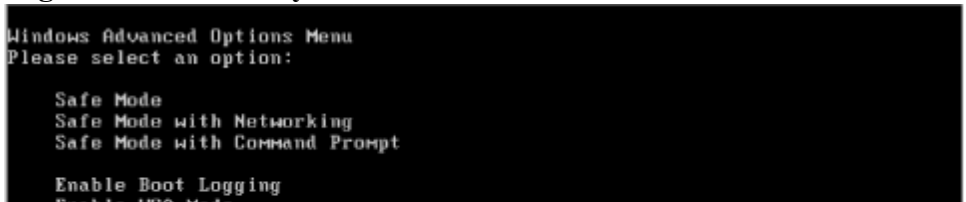
A nonauthoritative restore simply restores Active Directory without marking any of the data as authoritative. Since the data will be "nonauthoritative," any changes that have happened since the backup will replicate to the restored server. Also, any changes that were made on the server that had not replicated will be lost.

To perform a non-authoritative restore of a domain controller, you need to boot the DC into "Directory Services Restore Mode." The reason you have to do this is that when a domain controller is live, it locks the Active Directory database (*ntds.dit*) in exclusive mode. That means that no other processes can modify its contents. To restore over the DIT file, you must boot into DS Restore Mode, which is a version of Safe Mode for domain controllers. If you try to restore a live domain controller, you'll get an error like the one shown in Figure 13-4.

Figure 13-4. Restore error on a live domain controller

You can get into DS Restore Mode by hitting the F8 key during the initial system startup. After doing so you'll see the screen shown in Figure 13-5.

Figure 13-5. Directory Services Restore Mode



13.4 FSMO Recovery

The FSMO roles were described in Chapter 2. These roles are considered special in Active Directory because they are hosted on a single domain controller within a forest or domain. The architecture of Active Directory is highly redundant, except for FSMO roles. It is for this reason that you need to have a plan on how to handle FSMO failures.

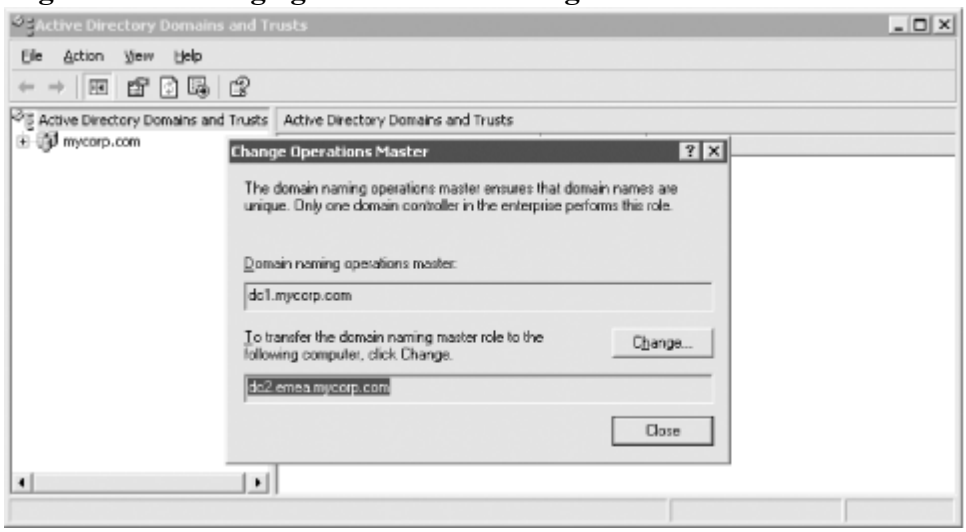
It would be a really nice feature if domain controllers could detect that they are being shut down and gracefully transfer any FSMO roles to other domain controllers. In fact, the Active Directory development team is considering this feature for the next major release of Active Directory after Windows Server 2003, but that is a ways out.

Without having the graceful FSMO role transfer, you have to do manual transfers. Manually transferring a role is pretty straightforward. You bring up the appropriate Active Directory snap-in, bring up the FSMO property page, select a new role owner, and perform the transfer. Here is a list of the FSMO roles and the corresponding snap-in that can be used to transfer it to another domain controller:

- Schema Master: Active Directory Schema
- Domain Naming Master: Active Directory Domains and Trusts
- RID Master: Active Directory Users and Computers
- PDC Emulator: Active Directory Users and Computers
- Infrastructure Master: Active Directory Users and Computers

Figure 13-9 shows the Active Directory Domains and Trusts screen for changing the Domain Naming FSMO.

Figure 13-9. Changing the Domain Naming FSMO role owner



When a FSMO role owner goes down and cannot be brought back online, you no longer can transfer the role; you instead have to "seize" it. And unfortunately you cannot seize FSMO roles using the Active Directory snap-ins as you can to transfer them. To seize a FSMO role you need to use the *ntdsutil* utility that we used earlier to do restores. We will now walk through the *ntdsutil* commands that are used to seize a FSMO role. Note that due to the width of the output, some of the text wraps to the following line.

We first start off by getting into the ntdsutil interactive mode and looking at the options for the roles command.

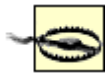
13.5 DIT Maintenance

On a periodic basis, such as a couple of times a year, you should check the health of the DIT file (*ntds.dit*) on your domain controllers. Using the *ntdsutil* utility, you can check the integrity and semantics of the Active Directory database and reclaim whitespace, which can dramatically reduce the size of the DIT. Also, just as you should rotate the password for the Administrator accounts in the forest, you should also change the DS Restore Mode Administrator password as well. You may even need to do this more frequently depending on whether you have people leave your team that should no longer know the password.

Unfortunately, to accomplish all these tasks—except changing the DS Restore Mode Administrator password—you have to boot the domain controller into DS Restore Mode. That means you will have to have schedule downtime for the machine. Also, to use DS Restore Mode, you need console access either through being physically at the machine or with out-of-band access, such as with Compaq's Remote Insight Lights Out Board (RILOE). There is one other option using Terminal Services. You can modify the *boot.ini* file on the domain controller to automatically start up in DS Restore Mode. You can then use a Terminal Services connection to log in to the machine. For more information, check out MS Knowledge Base article 256588 from http://support.microsoft.com.

13.5.1 Checking the Integrity of the DIT

There are several checks you can perform against the DIT file to determine whether it is healthy. The first we'll show checks the integrity of the DIT file. The integrity check inspects the database at a low level to determine whether there is any binary corruption. It scans the entire file, so depending on the size of your DIT file, it can take a while to complete. We've seen some estimates that state it can check around 2 gigabytes per hour, so allocate your change notification accordingly.



Before running any integrity checks, be sure you have at least two successful backups of the system.

To start the integrity check, run the *ntdsutil* command from within DS Restore Mode. The integrity subcommand can be found within the files menu.

```
 C:\> ntdsutil
ntdsutil: files
file maintenance: integrity
Opening database [Current].
Executing Command: C:\WINDOWS\system32\esentutl.exe /g"C:\WINDOWS\NTDS\ntds.dit" /o
Initiating INTEGRITY mode...
        Database: C:\WINDOWS\NTDS\ntds.dit
  Temp. Database: TEMPINTEG1752.EDB
Checking database integrity.
                    Scanning Status (% complete)
          0    10   20   30   40   50   60   70   80   90  100
          |----|----|----|----|----|----|----|----|----|----|
          ..................................................
Integrity check successful.
Operation completed successfully in 11.766 seconds.
Spawned Process Exit code 0x0(0)
If integrity was successful, it is recommended
 you run semantic database analysis to ensure
 semantic database consistency as well.
file maintenance: quit
```

The integrity check looks at the database headers to make sure they are correct and also checks all database tables to make sure they are working correctly. If the database integrity check fails or encounters errors, you may then want to run a repair command to try to fix the problem. Running an integrity check won't damage your Active Directory, but running a repair can, and that's why it is imperative you have a good backup before proceeding.

If the integrity check succeeds, you should then run a semantics check. Whereas the integrity check examines the

13.6 Summary

In this chapter we reviewed all the elements necessary to develop a disaster recovery plan. We covered how to back up Active Directory and some of the gotchas related to the tombstone lifetime and password change cycles. We then discussed the various options for restoring Active Directory, including restore by replication, authoritative restores, and nonauthoritative restores. We discussed the FSMO transfer process and what is needed to seize FSMO roles. Finally, we delved into some of the maintenance tasks that can be done with the Active Directory DIT files.

Chapter 14. Upgrading to Windows Server 2003

The first version of Active Directory with Windows 2000 was surprisingly stable and robust. Microsoft does not have the best track record for initial releases of products, but they must be commended for Windows 2000 Active Directory in terms of its feature rich-ness and reliability. That said, since Active Directory is such a complex and broad technology, there was still much room for improvement. There were some issues with scalability, such as the infamous 5,000-member limit with groups or the 300-site limit, which may have imposed artificial limitations on how you implemented Active Directory. Both of these issues have been resolved in Windows Server 2003. The default security setup with Windows 2000 Active Directory out-of-the-box was not as secure as it should have been. Signed LDAP traffic and other security enhancements have since been added into service packs, but they are provided by default with Windows Server 2003. Finally, manageability was another area that needed work in Active Directory, and in Windows Server 2003 numerous command-line utilities have been added along with some significant improvements to the AD Administrative snap-ins.

We have highlighted a few key areas where Active Directory has been improved in Windows Server 2003, and we'll describe more new features in the next section. If you already have a Windows 2000 Active Directory infrastructure deployed, your next big decision will be whether and when to upgrade to Windows Server 2003. Fortunately, the transition to Windows Server 2003 is evolutionary, not revolutionary, as with the migration from Windows NT to Active Directory. In fact, Microsoft's goal was to make the move to Windows Server 2003 as seamless as possible, and for the most part they have accomplished this. You can introduce Windows Server 2003 domain controllers at any rate you wish into your existing Active Directory environment; they are fully compatible with Windows 2000 domain controllers.

Before you can introduce Windows Server 2003 domain controllers, you must prepare the forest and domains with the ADPrep utility, which primes the forest for new features that will be available once you raise the functional level of the domain or forest. Functional levels are similar in nature to domain modes in Windows 2000 Active Directory. They allow you to configure different levels of functionality that will be available in the domain or forest based on which operating systems are running on the domain controllers.

Before we cover the upgrade process to Windows Server 2003, we'll first discuss some of the major new features in Windows Server 2003 and some of the functionality differences with Windows 2000. Based on this information, you should be able to prioritize the importance of how quickly you should start migrating.

14.1 New Features in Windows Server 2003

While the release of Windows Server 2003 is viewed as evolutionary, there are quite a few new features that make the upgrade attractive.

By "feature" we mean new functionality that is not just a modification of the way it worked in Windows 2000. In this sense, a feature is something you have to use or implement explicitly. Functionality differences with Windows 2000 are covered in the next section.

We suggest you carefully review each of these features and rate them according to the following categories:

1. You would use the feature immediately.
2. You would use the feature eventually.
3. You would never use the feature or it is not important.

Rating each feature will help you determine how much you could benefit from the upgrade. The following is the list of new features, in no particular order:

Application partitions

You can create partitions that can replicate to any domain controller in the forest.

Concurrent LDAP binds

Concurrent LDAP binds do not generate a Kerberos ticket and security token and are therefore much faster than a simple LDAP bind.

Cross-forest trust

This is a transitive trust that allows all the domains in two different forests to trust each other via a single trust defined between two forest root domains.

Domain controller rename

The rename procedure for domain controllers requires a single reboot.

Domain rename

Domains can now be renamed, but not without significant impact to the user base (e.g. all member computers must be rebooted twice). For more information, check out the following whitepaper: http://www.microsoft.com/windowsserver2003/downloads/domainrename.mspx.

Dynamic auxiliary classes

There is now support for the standards-based implementation of dynamic auxiliary classes. Under Windows 2000, auxiliary classes are considered "static" because they are statically defined in the schema. With dynamic auxiliary classes, you can link one when creating an object without it being defined in the schema as an auxiliary class for the object's objectClass.

Dynamic objects

Traditionally, objects are stored in Active Directory until they are explicitly deleted. With dynamic objects, you can create objects that have a time to live (TTL) value that dictates when they will be automatically deleted unless refreshed.

Install from media

14.2 Differences With Windows 2000

Even though Active Directory was scalable enough to meet the needs of most organizations, there were some improvements to be made after several years of real-world deployment experience. Many of the functionality differences with Windows 2000 are the direct result of feedback from AD administrators.

As with the new features, we suggest you carefully review each of the differences and rate them according to the following categories:

1. It would positively affect my environment to a large degree.
2. It would positively affect my environment to a small degree.
3. It would negatively affect my environment.

The vast majority of differences are actually improvements that translate into something positive for you, but in some situations, such as with the security-related changes, the impact may cause you additional work initially.

Single instance store

Unique security descriptors are stored once no matter how many times they are used as opposed to being stored separately for each instance. This alone can save upwards of 20%-40% of the space in your DIT after upgrading. Note that an offline defragmentation will have to be performed to reclaim the disk space.

Account Lockout enhancements

Several bugs have been fixed which erroneously caused user lockouts in Windows 2000. A new Active Directory Users and Computers property page called Additional Account Info and the *lockoutstatus.exe* utility are great troubleshooting tools for diagnosing lockout problems.

Improved event log messages

There are several new event log messages that will aid in troubleshooting replication, DNS, FRS, etc.

Link value replication (LVR)

Replication in Active Directory is done at the attribute level. That is, when an attribute is modified, the whole attribute is replicated. This was problematic for some attributes, such as the member attribute on group objects, which could only store roughly 5,000 members. LVR replication means that certain attributes, such as member, will only replicate the changes within the attribute and not the contents of the whole attribute whenever it is updated.

Intrasite replication frequency changed to 15 seconds

The previous default was 5 minutes, which has now been changed to 15 seconds.

No global catalog sync for PAS addition

With Windows Server 2003, whenever an attribute is added to the Partial Attribute Set (PAS), a global catalog sync is no longer performed as it was with Windows 2000. This was especially painful to administrators of large, globally dispersed Windows 2000 domains.

Signed LDAP traffic

Instead of sending LDAP traffic, including usernames and passwords, over the wire in plain text with tools such as ADUC and ADSI Edit, the traffic is signed and therefore encrypted.

ISTG and KCC scalability improvements

The algorithms used to generate the intersite connections have been greatly improved to the point where the previous limit of 300 to 400 sites has been raised to support roughly 3,000-5,000 sites.

Faster global catalog removal

14.3 Functional Levels Explained

Now that you are sufficiently excited about the new features with Active Directory and improvements since Windows 2000, we will now cover how you can actually enable these features in Windows Server 2003. If you've already deployed Windows 2000 Active Directory, you are most certainly familiar with the domain mode concept. With Windows 2000 Active Directory, you had mixed- and native-mode domains. Domain mode simply dictated what operating systems were allowed to run on the domain controllers and nothing more. New features were enabled with the move to native mode, including universal groups and group nesting to name a couple. Think of functional levels like domain modes, but taken a step further.

Windows Server 2003 functional levels are very similar to Windows 2000 domain modes from the standpoint that they dictate what operating systems can run on domain controllers, and they can only be increased or raised and never reversed. One common misunderstanding with domain modes, which hopefully will not be carried over to functional levels, is that they have virtually no impact on clients and what operating systems your clients run. For example, you can have Windows 9x clients in mixed- or native-mode Windows 2000 domains and also in domains that are at the Windows 2000 or Windows Server 2003 domain functional level.

For information about which operating systems are allowed at the various functional levels, check out Section 2.2.7 in Chapter 2.

An important difference with functional levels is that they apply both to domains and at the forest level. The reason for this is that some features of Windows Server 2003 Active Directory require either that all the domain controllers in a domain are running Windows Server 2003 or that all the domain controllers in the entire forest are running Windows Server 2003.

To illustrate why this is necessary, let's look at two examples. First, let's look at the new "Last logon timestamp attribute" feature. With this feature, a new attribute called lastLogonTimestamp is populated when a user or computer logs on to a domain, and it is replicated to all the domain controllers in a domain. This attribute provides an easier way to identify whether a user or computer has logged on recently than using the lastLogon attribute, which is not replicated and therefore must be queried on every domain controller in the domain. For lastLogonTimestamp to be of use, all domain controllers in the domain need to know to update it when they receive a logon request from a user or computer. Domain controllers from other domains only need to worry about the objects within their domain, so for this reason this feature has a domain scope. Windows 2000 domain controllers do not know about lastLogonTimestamp and do not update it. Therefore, for that attribute to be truly useful, all domain controllers in the domain should be running Windows Server 2003. All the domain controllers must know that all the other domain controllers are running Windows Server 2003, and they can do this by querying the functional level for the domain. Once they discover the domain is at a certain functional level, they start utilizing features specific to that function level.

Likewise, there are times when all domain controllers in the forest must be running Windows Server 2003 before a certain feature can be used. A good example is with the replication improvements. If some of the ISTGs were using the old site topology algorithms and others were using the new ones, you could have replication chaos. All domain controllers in the forest need to be running Windows Server 2003 before the new algorithms are enabled. Until then, they will revert to the Windows 2000 algorithms.

14.3.1 How to Raise the Functional Level

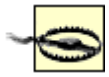
To raise the functional level of a domain or forest, you can use the Active Directory Domains and Trusts MMC snap-in. To raise the functional level of a domain, open the snap-in, browse to the domain you want to raise, right-click on it in the left pane, and select "Raise Domain Functional Level...". You will then see a screen similar to that in Figure 14-1.

Figure 14-1. Raising the domain functional level

14.4 Preparing for ADPrep

Before you can start enabling functional levels, you have to go through the process of upgrading your existing infrastructure to Windows Server 2003. The first step before you can promote your first Windows Server 2003 domain controller is to prepare the forest with the ADPrep utility.

If you've installed Exchange 2000 into your Active Directory forest, you are undoubtedly familiar with the Exchange *setup.exe* /forestprep and /domainprep switches. These switches are run independently from the Exchange server install to allow Active Directory administrators to take care of the AD-related tasks necessary to support Exchange. The Exchange /forestprep command extends the schema and adds some objects in the Configuration Naming Context. The Exchange /domainprep command adds objects within the Domain Naming Context of the domain it is being run on and sets some ACLs. The ADPrep command follows the same logic and performs similar tasks to prepare for the upgrade to Windows Server 2003.



Microsoft recommends that you have at least Service Pack (SP) 2 installed on your domain controllers before running ADPrep. SP 2 fixed a critical internal AD bug, which can manifest itself when extending the schema. There were also some fixes to improve the replication delay that can be seen when indexing attributes. If you plan on supporting a mixed Windows 2000 and Windows Server 2003 environment for an extended period of time, Microsoft recommends that you have SP 3 on your Windows 2000 domain controllers.

For more information on the Microsoft recommendations, check out Microsoft Knowledge Base Article 331161 from http://support.microsoft.com.

The ADPrep command can be found in the *\i386* directory on the Windows Server 2003 CD. The ADPrep command depends on several files in that directory so it cannot simply be copied out and put on a floppy or CD by itself. To run the ForestPrep, you would execute the following:

```
X:\i386\adprep /forestprep
```

where X: is a CD drive or mapped drive to a network share containing the Windows Server 2003 CD. Similarly, to run DomainPrep you would execute the following:

```
X:\i386\adprep /domainprep
```

You can view detailed output of the ADPrep command by looking at the log files in the *%SystemRoot%\system32\debug\adprep\logs* directory. Each time ADPrep is executed, a new log file is generated that contains the actions taken during that particular invocation. The log files are named based on the time and date ADPrep was run.

Now we will review what ForestPrep and DomainPrep do.

14.4.1 ForestPrep

The ADPrep /forestprep command extends the schema with quite a few new classes and attributes. These new schema objects are necessary for the new features supported by Windows Server 2003. You can view the schema extensions by looking at the *.ldf* files in the *\i386* directory on the Windows Server 2003 CD. These files contain LDIF entries for adding and modifying new and existing classes and attributes.

Microsoft warns against manually extending the schema with the ADPrep LDIF files. You should instead let ADPrep do it for you.

ForestPrep hardens some default security descriptors and modifies some of the ACLs on the containers in the Configuration NC. New displaySpecifier objects are added and some existing ones modified to support new features

14.5 Upgrade Process

The upgrade process to Windows Server 2003 should be straightforward for most deployments. No forest restructuring is required, no user profile or workstation changes are necessary assuming you are running the latest service pack and hotfixes, and there should be no need for political turf battles over namespace usage and ownership like there might have been with Windows 2000.

We are going to outline five high-level steps that you should follow to upgrade to Windows Server 2003. They include performing an inventory of your domain controllers and clients to determine if there will be any compatibility showstoppers. You are then ready to do a trial run and perform extensive testing to see what impact the upgrade may have on functionality. Next, you have to prepare your forest and domains with ADPrep, which we've already discussed in some depth. Finally, you'll upgrade your domain controllers to Windows Server 2003. In Section 14.6, we will describe what to do after you've upgraded your domain controllers as far as monitoring, raising functional levels, and taking advantage of new features goes.

14.5.1 Inventory Domain Controllers

A good first step before you start the upgrade process is to do a complete inventory of all of the hardware and software that is on your domain controllers. You'll then want to contact your vendors to determine whether they've already done compatibility testing and can verify support for Windows Server 2003. The last thing you want to do is start the upgrade process and find out halfway through that a critical monitoring application or backup software that runs on your domain controllers does not work correctly. Much of this testing can be done in your own labs, but it is always good to check with the vendors and get their seal of approval. After all, if a problem does arise, you'll want to make sure they are supporting the new platform and won't push back on you.

Next you'll want to ensure you have all the necessary hotfixes and service packs installed. A good overview of Microsoft's recommendations is documented in Microsoft Knowledge Base Article 331161. What you need to install depends on how long you plan on having your Windows 2000 domain controllers around. If you plan on a quick upgrade, you'll only need to do the minimal amount of patching required. But if you are going to have a prolonged migration, you should consider applying all the current fixes and service packs.

After you are sure that your hardware and software is fully up to date and will work under Windows Server 2003, you'll then want to do a very thorough check of your current domain controllers and make sure they are running without error. Go through the event logs and resolve any errors and warnings that may be occurring. The dcdiag and netdiag commands are useful for identifying potential issues. Also, if you don't already trend CPU and memory statistics, you'll need to start. The reason for collecting all this data is that if problems occur after the upgrade to Windows Server 2003, you'll want to narrow it down to whether it was previously a problem or if it is new, most likely as a result of the upgrade. If you don't collect this data, you are setting yourself up for trouble.

A good compatibility test is to run the /checkupgradeonly switch with the Windows Server 2003 installer (*winnt32.exe*).

```
X:\> i386\winnt32.exe /checkupgradeonly
```

This command will go through the steps as if you were upgrading, but it will check only the applications you have installed and the status of the forest. If you have not run ADPrep yet, it will return an error about that.

At this point you'll also want to check the status of your backups. Before you run ADPrep you should have successful backups for at least two domain controllers in every forest and every FSMO role owner. You should also ensure that your disasterrecovery procedures are well documented and have been tested.

14.5.2 Inventory Clients

The good news as far as clients go is that there aren't a lot of requirements for them to work in a Windows Server 2003 forest. In fact, there are no changes required for Windows XP and Windows 2000 machines. For NT 4.0 clients, you should have at least Service Pack 3, and Microsoft recommends Service Pack 6a. For Windows 98 and Windows 95 clients, they will need the DS Client installed as described in Microsoft Knowledge Base Article 323466

14.6 Post-Upgrade Tasks

After you've upgraded one or more of your domain controllers to Windows Server 2003, you need to do some additional tasks to fully complete the migration. First and foremost, you need to monitor the domain controllers every step of the way and especially after they have been upgraded. You are setting yourself up for failure if you are not adequately monitoring Active Directory.

14.6.1 Monitor

The criticality of monitoring cannot be overstated. If you are not monitoring, how can you determine whether something broke during the upgrade? Here are several things you should check after you upgrade your first domain controller in a domain, any FSMO role owner, and after all DCs have been upgraded:

Responds to all services

Query LDAP, Kerberos, GC (if applicable), and DNS (if applicable) and be sure authentication and login requests are being processed. The dcdiag command can run many of these tests.

Processor and Memory utilization

Trend processor and memory utilization for some period before you do the upgrade so you can compare to the numbers after the upgrade.

DIT growth

The growth of the DIT should not be significant. You may in fact want to do an offline defrag after the upgrade to reclaim any space due to single- instance store of ACLs.

Event logs

This is a no-brainer, but you should always check the event logs to see whether any errors are being logged.

DC resource records registered

Ensure that all of the SRV, CNAME, and A records for the domain controllers are registered. The dcdiag command can perform these checks.

Replication is working

Run repadmin /showreps and repadmin /replsum and watch for anything out of the ordinary.

Group Policies are being applied

You may want to add a new setting to an existing GPO or create a new GPO and see if the settings apply on a client that should be receiving it.

NETLOGON and SYSVOL shares exist

This can consist of opening an Explorer window and browsing the available shares on the domain controller.

FRS is replicating correctly

You can test this out by placing a test file in the SYSVOL share on a domain controller and waiting for it to replicate to the other domain controllers.

This is not a comprehensive list of everything you should possibly monitor, but it is a good start. If everything checks out over a period of a week, you can feel pretty comfortable that the upgrade was successful. If nothing else, as long as you keep a close eye on the event logs, you should be able to catch the majority of problems.

14.6.2 Raise Functional Levels

After you feel comfortable that the upgrades have completed successfully, your next step should be to start raising the functional levels. If you've only upgraded the domain controllers in a single domain, you can raise the functional level for only that domain to Windows Server 2003. If you've upgraded all the domain controllers in the forest, you can also proceed to upgrade the forest functional level to Windows Server 2003.

14.7 Summary

In this chapter, we covered the new features in Windows Server 2003 and some of the differences with Windows 2000, most of which were instigated by real-world deployment issues. We then went over how you can enable new features with the use of functional levels and why they are necessary. Next we discussed the ADPrep process and how that must be done before the first Windows Server 2003 domain controller can be promoted. Once you have your forest and domains prepared, you can start the upgrade process. We described some of the important issues to be aware of when upgrading, and finally what to do after you've completed the upgrade.

While this chapter focused mainly on upgrading from an existing Windows 2000 Active Directory infrastructure, in the next chapter we discuss some of the key issues with migrating from Windows NT straight to Windows Server 2003 Active Directory.

Chapter 15. Migrating from Windows NT

Knowing how to design Active Directory is very useful, but it's not the end of the story. You may already have an existing NetWare or Windows NT infrastructure and want to consider migrating to Active Directory. Alternatively, you may have existing directories and networks that you would like Active Directory to complement rather than replace. One of the most important features of Active Directory is its ability to integrate with other directory services.

In this chapter we will cover some of the issues to consider when migrating from a Windows NT environment to Active Directory. Migrating to Active Directory from an existing NOS infrastructure is analogous to jumping from one moving car to another. This is due to the fact that organizations rarely get the opportunity to take extended downtime from both the client and server perspective to move everyone to Active Directory. In fact, limiting downtime for users is typically one of the top prioritie, so having a well-thought-out migration and fallback plan is critical to reduce the impact to your user base.

15.1 The Principles of Upgrading Windows NT Domains

There are many reasons that you will want to upgrade your Windows NT domains to Active Directory, not least of which is to make use of Active Directory and other features. It's possible to have significantly fewer domains in Active Directory because each domain can now store millions of objects. While fewer domains means less administration, the added benefit of using organizational units to segregate objects makes the Active Directory represent a business more accurately, both geographically and organizationally, and is a significant step forward. Couple this with the ability to set ACLs on objects and their properties in Active Directory, and you get much more fine-grained control for administrative delegation than before. You also can start phasing out old services, such as Windows Internet Naming Service (WINS) and extraneous Windows NT Backup Domain Controller (BDC) servers, since the clients now make more efficient use of DCs via TCP/IP and DNS. With all these improvements, the goals of upgrading a domain are easy to state:

- Reduce the number of domains in use since it is easier to administer fewer domains.
- Gain an extensible schema that allows much more corporate information to be stored than was previously possible.
- Create a hierarchical namespace that as closely as possible mirrors the organizational structure of the business.
- Gain much more fine-grained control over delegation of administration without needing to resort to the use of multiple domains.
- Reduce network bandwidth use by DCs through both multimaster replication and a significantly more efficient set of replication algorithms.
- Reduce the number of PDCs/BDCs to a smaller number of DCs through a more efficient use of DCs by clients.
- Eliminate the need for reliance on WINS servers and move to the Internet-standard DNS for name resolution.

To get the maximum benefit from the new technologies, you really need to upgrade both clients and servers.

15.1.1 Preparing for a Domain Upgrade

There are three important steps in preparing for a domain upgrade:

1. Test the upgrade on an isolated network segment set aside for testing.
2. Do a full backup of the SAM and all core data prior to the actual upgrade.
3.

15.2 Summary

This chapter focused on the principles behind the migration of existing Windows NT domains to Active Directory. Microsoft has taken the time to properly think through a very scalable and stable directory service in its Active Directory implementation. It has, in its own words, "bet the barn on Active Directory."

The next chapter takes a look at the potential for integrating Microsoft Exchange into Active Directory.

Chapter 16. Integrating Microsoft Exchange

Exchange 2000 has been the driving reason behind many companies' move to Active Directory. Exchange 2000 requires an Active Directory infrastructure, and the dependencies it places on AD are not small. In fact, the Exchange 2000 schema extensions roughly double the size of the default Active Directory schema. There are also restrictions on the location of your domain controllers relative to the Exchange servers. For these reasons and the critical nature of email, calendar, and collaboration services, all of which Exchange can provide, it is clear that Exchange 2000 can be the most significant application you integrate into Active Directory.

In this chapter, we will briefly touch on some of the important issues regarding the integration of Exchange with Active Directory. We'll cover how to prepare the forest for Exchange and describe some of the changes this causes. Finally, we will review the Active Directory Connector (ADC), which aids in the transition from Exchange 5.5 to Exchange 2000.

16.1 Quick Word about Exchange Server 2003

Exchange Server 2003, the next major release of Exchange, is currently due out in the summer of 2003. While there are many new features planned for that release, the way it integrates with Active Directory largely remains the same. This chapter focuses on Exchange 2000, but the concepts and procedures we describe map very closely to Exchange Server 2003 as well.

Here are a few key points to note about Exchange Server 2003 and Windows Server 2003:

- Exchange 2000 can only run on Windows 2000.
- Exchange Server 2003 can run on Windows 2000 and Windows Server 2003.
- Exchange 2000 can run in a Windows Server 2003 or Windows 2000 Active Directory forest.
- Exchange Server 2003 can run in a Windows Server 2003 or Windows 2000 Active Directory forest.
- Exchange 5.5 can interoperate with Exchange Server 2003 and Windows Server 2003 just as it could with Exchange 2000 and Windows 2000.
- The Outlook 2003 mail client allows cross-forest authentication with Windows Server 2003 forests.

16.2 Preparing Active Directory for Exchange 2000

Before you can install the first Exchange 2000 server in Active Directory, you have to prepare your forest. The Exchange setup program provides two options called /forestprep and /domainprep, which perform various tasks such as extending the schema, creating groups, creating containers for Exchange, and setting permissions on those containers. Due to the extent of changes caused by running these commands and the elevated privileges required to do so, it is imperative that AD administrators have a thorough understanding of what they do.

16.2.1 Forestprep

The Forestprep option of the Exchange 2000 setup extends the schema and makes some changes to the Configuration container. Forestprep must be run before Domainprep can be executed and subsequently before you can install your first Exchange 2000 server. The user that runs Forestprep must be a member of both the Enterprise Admins and Schema Admins groups. Here is a list of some of the tasks Forestprep takes care of:

- Extends the schema with close to 2000 schema additions and modifications. Forestprep effectively doubles the number of classes and attributes in the default Active Directory schema. Several attributes are also added to the Global Catalog, which will cause a GC resync with Windows 2000 Active Directory.
- Creates the Exchange organization with the following distinguished name: `cn=<ExchangeOrgName>,cn=MicrosoftExchange,cn=Services,cn=Configuration,<ForestDN>`.
- This container is where Exchange stores most of its data in Active Directory, including the address lists, administrative groups, recipient policies, and other global settings.
- Grants full control rights to the designated user or group over the Exchange organization. The rights granted are equivalent to the Exchange Full Administrator rights when using the Exchange Delegation of Control wizard.

Due to the massive number of schema extensions, you should consider running Forestprep on the Schema FSMO role owner. This can speed up the time it takes for complete Forestprep. Before moving forward to Domainprep, you must ensure that the schema extensions and objects injected by Forestprep have replicated across the forest.

16.2.2 Domainprep

After you've successfully run Forestprep, you need to run Domainprep in any domain in which you plan to install an Exchange 2000 server or have mail-enabled users. The user that runs Domainprep must be a member of the Domain Admins group for the target domain. Some of the tasks performed during Domainprep include the following:

- Creates a container for the System mailboxes under cn=Microsoft Exchange System Objects, *<DomainDN>*
- Creates the Exchange Domain Servers global group, which is the default location for new Exchange 2000 servers in the domain.
- Creates the Exchange Enterprise Servers domain local group. The Recipient Update Service eventually adds all the Exchange Domain Servers groups from each domain to this group.
-

16.3 Exchange 5.5 and the Active Directory Connector

A lot of companies that are migrating to Exchange 2000 had Exchange 5.5 deployed previously. To help with the transition process, Microsoft created the Active Directory Connector (ADC), which allows you to migrate at your own pace while maintaining both environments.

The ADC is comprised of a service that does the work and an MMC console to manage the service. While the console can be installed on any client or server, the ADC service has to be installed on a DC for it to work.

To support connection to the ADC, you will need Microsoft Exchange 5.5 Service Pack 1 or above.

When you install the ADC for the first time in a forest, it extends the schema to include new Exchange objects and attributes, as well as modifying existing Active Directory objects to include new Exchange-relevant attributes. The Exchange Schema is also modified if you intend to replicate Active Directory data to Exchange. For example, the User class object in the Active Directory Schema is directly modified to include three Exchange-relevant auxiliary classes in the auxiliary class attribute: msExchMailStorage, msExchCustomAttributes, and msExchCertificateInformation. Auxiliary classes and schema are discussed more fully in Chapter 4.

Once the Active Directory schema is extended, Active Directory then can hold mail attributes for groups, users, and contacts just as the Exchange directory can. This means that the ADC now can replicate data bidirectionally, knowing that either end can store the same data. This allows you to run the ADC in one of three ways:

From Active Directory to Exchange

Every new creation of a user, distribution group, security group, or contact object that is mail-enabled in a designated Organizational Unit will be copied over to a designated Recipients container on Exchange. Every change to the attributes of an existing mail-enabled object will also be passed. Deletions also can be synchronized.

From Exchange to Active Directory

Every new creation of a user, mailing list, or contact object in Exchange automatically creates a corresponding user account in a specified Organizational Unit in Active Directory. Attribute changes also get passed, as do deletions.

Bidirectional replication

Changes at either end get replicated over to the other system.

If you choose to manage one-way replication, you must appreciate that you can update the details only for those objects on the one-way source directory from that time on. If you were to update the target directory, the changes you made could potentially be erased during the next update as the system realizes that the target is no longer in synchronization with the source. To fully appreciate this and see why bidirectional replication does not necessarily help you here, see the later Section 16.3.2 and Section 16.3.3.

There are other implications that need to be understood for these scenarios. When passing information from Active Directory to Exchange, for example, you must designate a set of specific Organizational Units that will contain the objects to be replicated. Any Organizational Units that you do not list will never have objects replicated, even if they are mail-enabled objects.

Once the ADC is installed, the Active Directory Users and Computers MMC has three extra property pages available to it. Two of these pages are visible only if you choose the Advanced option from the View menu. One word of warning: to see the extra pages in the Active Directory Users and Computers MMC on any server or workstation, you must have the ADC MMC installed onto that client first. Installing the MMC part of the ADC onto a client configures the Active Directory User and Computers MMC with the extra snap-in options for these pages.

We'll now take a look at how to configure the ADC for your use and follow on with how to mail-enable a user using the GUI and ADSI.

16.4 Summary

The importance of Exchange 2000 in the enterprise is ever increasing. Exchange has steadily eaten away at the messaging market to the point where it is currently the market leader. In fact, the initial driving force behind the move to Active Directory for many organizations is the need to deploy Exchange 2000. Integrating Exchange into Active Directory is no small feat due to its heavy reliance on AD. For companies migrating from Exchange 5.5, the Active Directory Connector (ADC) can help in the transition, but it introduces additional support overhead.

While Exchange 2000 can be the most significant application you'll integrate with Active Directory, it is by no means the only one you can or should integrate. In the next chapter, we will dive into more details around the future of Microsoft's Directory Services strategy and how that impacts integration of applications with Active Directory.

Chapter 17. Interoperability, Integration, and Future Direction

Microsoft's Directory Services strategy has come a long way in the past few years. Even before Active Directory, several Microsoft products utilized a directory, although most used one that was built in. Some examples include the NetMeeting ILS server and Exchange 5.5, which was the precursor to Active Directory. With the introduction of Active Directory in 1999, Microsoft finally had the first signs of a coherent Directory Services strategy. With the release of Windows Server 2003, plus a major overhaul of Microsoft Metadirectory Server and the introduction of Active Directory Application Mode, Microsoft has one of the most diverse and robust directory offerings of any of the major directory vendors in the market today. In this chapter, we will discuss Microsoft's future plans for Directory Services and cover how that plan fits in with interoperating with other directories and integrating with applications and services.

17.1 Microsoft's Directory Strategy

After the initial release of Active Directory, Microsoft thought, like many in the industry, that the direction most companies were headed was deployment of a single enterprise directory that would be all things to all clients. Microsoft's intent was for Active Directory to serve the NOS directory role, replacing NT 4.0, and also the application directory role, which had typically been dominated by SunOne (formerly iPlanet) and OpenLDAP. But after three years of implementations, it became evident that although most companies would like to implement a single directory, in practice it did not work out that way. A lot of applications are developed with a particular directory in mind and in some cases, like Exchange 2000, an application can work only with a specific directory. After Microsoft realized that multiple directories would be a reality in most organizations of any size, they decided to rework their strategy. This happened to coincide with their plans to release a major update of the Windows Server operating system, Windows Server 2003.

There are three main components to Microsoft's current Directory Services roadmap: Active Directory Application Mode (AD/AM) as the application directory, Microsoft Metadirectory Services (MMS) as the central provisioning source, and Active Directory as the NOS or Infrastructure directory. We'll now examine each of these products.

17.1.1 Active Directory Application Mode

When Microsoft announced plans in July 2002 to release a "lightweight" version of Active Directory sometime after the release of Windows Server 2003, many AD administrators breathed a sigh of relief. The reason for the relief is that when Active Directory serves as a NOS directory, as it does in the vast majority of implementations, it does not lend itself well to being a flexible application directory. We describe some of the challenges of using Active Directory in both roles in Section 17.3, later in the chapter.

Active Directory Application Mode, or AD/AM for short, will help reduce the need for Active Directory to serve dual purposes. AD/AM is closer to what most consider a traditional LDAP directory, such as that offered by SunONE and OpenLDAP. It has many AD-specific features stripped out, such as KDC support and DNS SRV requirements, which are necessary for the DC Locator process. AD/AM actually uses the same code base as Active Directory, but the unwanted features are disabled. Some of the similarities with Active Directory include:

- Support for many of the same tools (e.g., ADSI Edit and LDP)
- Support for ADSI and LDAP
- Support for multimaster replication
- Support for a fully extensible schema, although a very minimal schema is provided out of the box
- Inclusion of Configuration, Schema, and Application Partitions (but no Domain Partitions)

Some of the differences from Active Directory include:

- Easy setup process (not dcpromo) with no reboot required
- Support for installing multiple instances on a single computer
- Capability to run each instance as a service and to stop and start services without a reboot

17.2 Interoperating with Other Directories

Now that we've covered what Microsoft is doing with their directory products, let's review some of the issues around integrating a mixed directory environment. As we mentioned earlier, supporting multiple directories within a large organization is a necessary practice. You may already have several directories deployed, some of which are not Microsoft-based. A common question in this scenario is how to get your directories to work together.

17.2.1 Getting Data from One Directory to Another

Perhaps the most common use of a directory is to access employee, customer, or student information. One of the problems of supporting multiple directories is that for each directory to be useful, it needs to store similar data. It would be very helpful if there were a standard RFC that defined a replication scheme for LDAP directories, but unfortunately there is not. As a result, each directory vendor has implemented their own way to replicate data between servers. This is where metadirectories come into play. The primary purpose of a metadirectory is to facilitate data flow and provisioning across systems. If you have several directories, and writing your own scripts to replicate data is not a possibility, implementing a metadirectory is a valid option.

17.2.2 Using Common Tools Across Directories

One of the biggest reasons for not wanting to implement multiple directories is that they have to be managed differently. Fortunately, both Active Directory and AD/AM are based on LDAP, so any of the standard LDAP SDK tools such as ldapsearch and ldapadd will work. Also, the Microsoft LDP tool, a graphical user interface for querying and managing content in Active Directory, has become very popular. LDP is an LDAP-based tool and works against any LDAP directory. The same cannot be said for tools such as ADSI Edit and the Active Directory administrative snap-ins, which works only with Active Directory.

One popular approach for managing content in SunONE and OpenLDAP directories is to use the LDAP Data Interchange Format (LDIF). LDIF has a strict format that is both human- and machine-readable, but it is easy to work with. Microsoft provides the LDIFDE program on the Windows Server platforms, which allows for importing and exporting LDIF files. You can also use an LDIF-based tool on a non-Windows platform to manage content in Active Directory.

17.2.3 Porting Scripts to Work Across Directories

The story for porting scripts is much the same as the one for using similar tools for managing different directories. Most directories today are LDAP-based, so if your scripts are using an LDAP API, they should work regardless of what directory is being used. That said, there are some fairly significant differences with how Active Directory was implemented that may cause problems in your scripts. Most LDAP directories, including AD/AM, have a flat namespace. That means you can make a single query to a server and retrieve all objects the server knows about. With Active Directory, it is a little different in multidomain environments. When you implement multiple domains, you are essentially segregating your LDAP namespace. A domain controller knows about only the objects in its domain. For this reason, Microsoft designed the Global Catalog so that you can perform a single query to search against all objects in a forest, but the GC contains only a subset of information for all objects. The impact to scripts may be less than obvious, but to perform a query such as retrieving all attributes for any user in the forest that has a department equal to "Sales", you first must query the GC. To then retrieve all defined attributes for each user, you have to run separate queries against the domains the users are in. The other option is to skip the GC and query the domains individually, but regardless this simple task can require several queries.

17.2.4 Making Searches Across Directories Seamless

If you foresee supporting multiple directories, you might have the notion of trying to unify the namespace used by each. So perhaps your Active Directory root is dc=mycorp,dc=com and you have an OpenLDAP server that has a root at dc=apps,dc=mycorp,dc=com. You can create referral objects using the crossRef objectclass so that a query for dc=apps,dc=mycorp,dc=com against an Active Directory domain controller will refer the client to an OpenLDAP server. The LDIF representation of the referral object looks like the following, where nCName is the name of the

17.3 Integrating Applications and Services

Many applications rely on a directory to access user information and store application data. Since Active Directory was Microsoft's first true directory offering, many application vendors attempted to integrate their products into it, only to find there were a lot of issues from both technology and political perspectives. We'll now discuss some of these challenges.

17.3.1 The Application Integration Challenge

While trying to use Active Directory as both a NOS and application directory can initially reap significant rewards from reduced total cost of ownership, it also presents several challenges as well. In fact, many of the features that make Active Directory a great NOS directory (a repository of user, group, and computer accounts) also make integrating applications much more difficult.

17.3.1.1 Challenges for application vendors

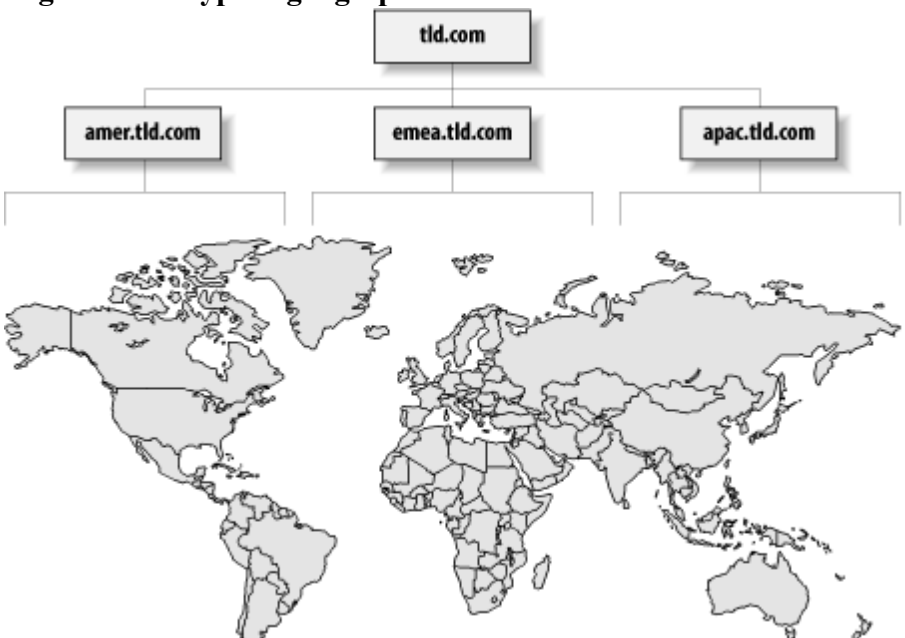
Many of the challenges for application vendors are related more to incompatibilities with integrating with the NOS than directly to insufficiencies with Active Directory. In fact, Active Directory could be used as a pure application directory with few differences from what you would see using a SunONE or OpenLDAP directory server. But that is not how Active Directory is typically being used in the enterprise. In fact, most organizations are still trying to balance the effects of maintaining a stable NOS environment that has consistent reliability and response times with an application directory that could impact the end-user experience with increased server load and directory bloat.

We have seen numerous vendors struggle with trying to integrate products with Active Directory, especially on their first attempt. Most companies do not have a lot of Active Directory or even LDAP expertise, so they make do with what they have, which often results in poorly integrated applications. In fact, it is not sufficient for vendors to have just LDAP expertise, because Active Directory has many features, such as the Global Catalog, never seen in any other directory server product. Often vendors gain the expertise they need only after they have struggled through the painful experiences of several customers that have deployed their product. Some of the major issues application vendors face are described in the following list.

Hierarchical structure

One of the biggest roadblocks for applications using Active Directory is accessing data in a multidomain model. Most medium- to largescale Active Directory implementations use multiple domains to segregate data, regulate administrative access, limit exposure during disaster recovery situations, and reduce the amount of data that replicates between domain controllers. Typically, the domains are spread across geographic and sometimes organizational boundaries. Figure 17-1 illustrates an example of a simple geographic domain structure that is commonly used.

Figure 17-1. Typical geographic domain model



17.4 Summary

Integrating applications into Active Directory is not an easy task. There are several potential pitfalls not only for Active Directory administrators but for application developers as well. Active Directory Application Mode (AD/AM), which is a lightweight version of Active Directory, should help relieve some of the burden from Active Directory as an application directory. Integrating Unix with Active Directory also has its challenges, but it is possible.

While integrating applications can be a challenge, getting competing directory services to interoperate is downright difficult. Even though most directory servers are based on standards-based RFCs, such as LDAP, there are no standards that define how they can replicate or authorize seamlessly with each other. The two best options for integrating multiple directories is either through a metadirectory, such as MMS, or a programming interface, such as ADSI or LDAP.

This concludes Part II. In Part III, we will cover many of the programmatic concepts and interfaces that can be used to automate and manage your Active Directory environment.

Part III: Scripting Active Directory with ADSI, ADO, and WMI

In the networks of today, companies can have tens of thousands of users on hundreds of servers in an organization that spans many sites. Managing complex systems can take a lot of time, and setting up the mechanisms to effect sensible management can be cumbersome.

Windows Server 2003 and Windows 2000 provide the administrator with a variety of tools to manage Active Directory. Unfortunately, these tools are no help for a variety of tasks that you may need to do en masse. No one in his right mind creates thousands of user accounts using the Active Directory Users and Computers snap-in. You can also manage and manipulate the Active Directory objects using scripts—and very powerful scripts at that. You can write scripts to manipulate any object and its properties, and you can port these scripts to the web, allowing administration through a browser interface.

Before we start, we want to state categorically that scripting Active Directory is easy. You don't have to know complex code algorithms, pointer structures, object class inheritance, or any of the weird world of complex program languages. Here we use Microsoft's VBScript language, a very simple language both to use and to understand. You should have no problem coming to this section with zero knowledge and being able to understand and implement the concepts behind the chapters in the section.

Chapter 18. Scripting with ADSI

This chapter covers the basics of ADSI and VBScript so that even inexperienced programmers and system administrators can understand how to write useful scripts. If you're used to another language, such as VB, you'll find that it is very easy to convert the ADSI examples from VBScript, which is covered in detail in Chapter 25. In Chapter 25 we also cover how to add VBScript code to HTML web pages so that you can write web applications that utilize ADSI. In Chapter 20, we show you how to use ADO to search Active Directory and retrieve sets of records according to the powerful search conditions that you impose. Other chapters take this knowledge and extend it so that you can manipulate other aspects of Active Directory, such as permissions and auditing (Chapter 23) and modifying the schema (Chapter 24). Several additional references to web pages containing further information and documentation are included at the end of this chapter, so that you can find more information.

18.1 What Are All These Buzzwords?

First, let's take a look at some of the underlying technologies that you'll use when developing scripts.

18.1.1 ActiveX

ActiveX, the base component of a number of these technologies, enables software components to interact with one another in a networked environment, regardless of the language in which they were created. Think of ActiveX as the method developers use to specify objects that the rest of us then create and access with our scripts in whatever language we choose. Microsoft currently provides three hosts that run scripts to manipulate ActiveX objects: the Internet Information Server (IIS) web server, the Internet Explorer (IE) web browser, and the Windows Scripting Host (WSH). IIS allows scripts called from HTML pages to run on the host server, and IE runs scripts called from HTML pages on the client. WSH allows scripts to run directly or remotely on a host from a command-line or GUI interface. WSH is an integral part of the Windows operating system.

18.1.2 Windows Scripting Host (WSH)

WSH is an important technology for a number of reasons:

- You need no other software to start scripting.
- The development environment for WSH has no special requirements to build or compile programs; your favorite text editor will do.
- You can execute any WSH script with a VBS, JS, or WSF extension just by double-clicking it.
- You can actually execute scripts from the command line, directing window output to that command line. This is possible because WSH has two interpreters, one called *wscript.exe*, which interprets scripts in the GUI Windows environment, and one called *cscript.exe*, which interprets scripts in the command-line environment of a *cmd.exe* session. By default, if you double-click a script called *myscript.vbs*, the system passes that script to *wscript.exe*, just as if you had manually typed wscript.exe myscript.vbs. The default interpreter can be changed generally or on a per-script basis along with other settings.
- WSH comes with a series of procedures that allow you to script interactions with the target machine. There are procedures for running programs, reading from and writing to the registry, creating and deleting files and shortcuts, manipulating the contents of files, reading and writing environment variables, mapping and removing drives, and adding, removing, and setting default printers. These procedures are native to WSH, meaning that only scripts executing under WSH can access them. Being able to access these settings is very useful when configuring users' environments, since you can now write logon scripts using VBScript or JScript if you wish.

WSH comes bundled with Windows Server 2003, Windows XP, Windows 2000, and Windows 98, and it can be downloaded from http://www.microsoft.com/msdownload/vbscript/scripting.asp and installed on Windows 95 and Windows NT 4.0 servers and workstations.

18.1.3 Active Server Pages (ASPs)

When a VBScript is wrapped inside an HTML page, it is called an Active Server Page (ASP) because it can contain dynamic (or active) content. This means that the web page displayed to the user differs depending on the results of a

18.2 Writing and Running Scripts

The third part of this book is dedicated to showing you techniques to access and manipulate Active Directory programmatically. It not only contains a plethora of useful scripts that you will be able to adapt for use in your organization, but it also contains a lot of information on how you can write your own scripts to access Active Directory to do whatever you need. Let's take a quick look at how to get started writing and running scripts.

18.2.1 A Brief Primer on COM and WSH

Since the release of Windows 2000, each operating system Microsoft has produced comes with a technology called the Windows Scripting Host, more commonly known as WSH, which allows scripts to execute directly on the client. WSH-based scripts can open and read files, attach to network resources, automate Word and Excel to create reports and graphs, automate Outlook to manipulate email and news, change values in the registry, and so on. The reason these scripts can be so versatile is that WSH supports scripting access to all Component Object Model (COM) objects installed on the client.

COM is a Microsoft technology that allows programmers to automate and manipulate virtually anything you require on a host by defining each host component as a set of objects. When someone needs to create or manage a new component on a Windows-based host, she creates a COM interface, which can be thought of as the definition of the object and the entire set of operations that can be performed on that object. Interfaces normally are stored in DLL files.[1]

[1] There are other file types, such as OCX controls that define graphical forms and windows you can use in your scripts, but they are beyond the scope of this book.

For example, if you want to manipulate a file, you actually need to manipulate a file COM object. The file COM object definition is stored in an interface held in a DLL. The interface also holds all of the operations, such as creating the file, deleting the file, writing to the file, and so on. The interface also defines a series of properties of the object, such as the filename and owner, which can be accessed and modified. Procedures that operate on an object are known as methods, whereas the properties of an object are known simply as properties.

In addition to methods and properties provided by interfaces, each scripting language that you use has a series of defined functions, such as writing to the screen or adding two numbers together.

You can write scripts that execute using WSH and access any COM objects available to you using the methods and properties defined in the interface for that object and any functions in your chosen scripting language. By default, you can use Microsoft VBScript or Microsoft JScript (Microsoft's version of JavaScript). WSH is fully extensible, so other language vendors can provide installation routines that update WSH on a client to allow support for other languages. A good example is PerlScript, the WSH scripting language that provides support for the Perl language.

18.2.2 How to Write Scripts

WSH scripts are simple to write. The following example is a very simple script written in VBScript and called *simple.vbs*:

```
MsgBox "Hi World!"
```

All you have to do is open up your favorite text editor type in the command, then save the file with a specific filename extension (VBS for VBScript or JS for JScript). Then you can double-click the script and it will run using WSH. Figure 18-1 shows the output of the script, which is a simple dialog box with a text string in it. The script uses the VBScript MsgBox function.

Figure 18-1. Output from a very simple script

18.3 ADSI

Before you can start writing scripts that use ADSI, you first need to understand the basic COM concept of interfaces and ADSI's concepts of namespaces, programmatic identifiers (ProgIDs), and ADsPaths.

18.3.1 Objects and Interfaces

A COM interface defines the properties associated with an item, how to access those properties, and how to access specific functionality of the item, more commonly referred to as an object. For example, WSH has a number of objects that represent files, shortcuts, network access, and so on. ADSI provides a specification for interfaces that each directory service provider must implement to maintain uniformity. Each ADSI interface normally supports methods that can be called to perform a specific action, and properties (or property methods) to retrieve information about the object.

A method is a procedure or function that is defined on an object and interacts with the object. So an interface to access Active Directory group objects would have Add and Remove methods, so that members could be added or removed from a group. Methods are normally represented as Interface::MethodName when referenced, and this is the form we adopt in this book. Objects also have properties that are retrieved using the IADs::Get or IADs::GetEx methods and set or replaced using the IADs::Put or IADs::PutEx methods.

Each ADSI object supports an IADs interface that provides six basic pieces of information about that object:
Name

Relative name for the object (RDN in the case of Active Directory)
ADsPath

Unique identifier for object
GUID

128-bit Globally Unique Identifier of object
Class

Objectclass of the object
Schema

ADsPath to the objectclass of the object
Parent

ADsPath to the parent object

If you wanted to retrieve the GUID property of an object, you would use the following:

```
strGUID = objX.Get("GUID")
```

You can see that we are calling the IADs::Get method on the object called objX; the dot (.) indicates the invocation of a property or method. The IADs::Get method takes as its one parameter the property to retrieve, which in this case is the GUID, and passes it out to a variable that we have called strGUID. So that you do not have to use the IADs::Get method for the most common properties, certain interfaces define these common properties with property methods. In these specific cases, you use the dotted method notation to retrieve the property by using the property method of the same name. In the previous GUID example, the GUID property has a property method of the same name (i.e., IADs::GUID). We could therefore retrieve the GUID with:

```
strGUID = objX.GUID
```

We won't go into the interfaces in any more depth here; we just want to give you a feel for the fact that methods and properties can be accessed on an object via ADSI interfaces. Although an object can support more than one interface without a problem, each object supports only the interfaces that are relevant to it. For example, the user object does not support the interface that works for groups. The other interfaces, of which there are around 40, begin with the

18.4 Simple Manipulation of ADSI Objects

Let's now take a look at simple manipulation of Active Directory objects using ADSI. We are using Active Directory as the primary target for these scripts, but the underlying concepts are the same for any supported ADSI namespace and automation language. All the scripts use GetObject to instantiate objects, assuming you are logged in already with an account that has administrator privileges; if you aren't, you need to use IADsOpenDSObject::OpenDSObject as shown earlier in the chapter.

The easiest way to show how to manipulate objects with ADSI is through a series of real-world examples, the sort of simple tasks that form the building blocks of everyday scripting. To that end, imagine that you want to perform the following tasks on the *mycorp.com* Active Directory forest:

1. Create an Organizational Unit called Sales.
2. Create two users in the Sales OU.
3. Iterate through the Sales OU and delete each user.
4. Delete the Organizational Unit.

This list of tasks is a great introduction to how ADSI works because we will reference some of the major interfaces using these examples.

18.4.1 Creating the OU

The creation process for the Sales Organizational Unit is the same as for any object. First you need to get a pointer to the container in which you want to create the object. You do that using the following code:

```
Set objContainer = GetObject("LDAP://dc=mycorp,dc=com")
```

While VBScript and VB have the GetObject function, VC++ has no such built-in function. ADSI provides the ADsGetObject function for use by those languages that need it.

Since we are creating a container of other objects, rather than a leaf object, you can use the IADsContainer interface methods and properties. The IADsContainer::Create method is used to create a container object, as shown in the following code:

```
Set objSalesOU = objContainer.Create("organizationalUnit","ou=Sales")
```

Here we pass two arguments to IADsContainer::Create: the objectclass of the class of object you wish to create and the Relative Distinguished Name (RDN) of the object itself. We use the ou= prefix because the type of object is an Organizational Unit. Most other objects use the cn= prefix for the RDN.

The IADsContainer interface enables you to create, delete, and manage other Active Directory objects directly from a container. Think of it as the interface that allows you to manage the directory hierarchy. A second interface called IADs goes hand in hand with IADsContainer, but while IADsContainer works only on containers, IADs will work on any object.

To commit the object creation to Active Directory, we now have to call IADs::SetInfo:

```
objSalesOU.SetInfo
```

ADSI implements a caching mechanism in which object creation and modification are first written to an area of

18.5 Further Information

This is by no means an in-depth discussion on ADSI. For more information, you should look at the Microsoft Developer Network (MSDN) library documentation, which contains all of the documentation on the specifics of VBScript, JScript, ADO, ADSI, and WSH. There are a few ways to get hold of the MSDN library: you can purchase an MSDN library subscription from Microsoft and get quarterly CDs with all of the documentation, or you can access the documentation directly via the Internet. MSDN online can be found at http://msdn.microsoft.com/library/. Once you enter the MSDN library from the CD-ROM or the Web, you will see a list of contents on the left-hand menu, which you can browse.

Table 18-1 lists some useful Internet sites to find additional information on the topics covered in this chapter.

Table 18-1. Useful Internet sites

Description	**URL**
Microsoft's main scripting web site	http://msdn.microsoft.com/scripting/
MSDN Library root	http://msdn.microsoft.com/library/
WSH docs	http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001169
Microsoft's universal data access components site (including the official pages for ADO)	http://www.microsoft.com/data/
A fantastic site for developers of ASP, ADSI, and ADO pages and scripts (including a superb ADSI mailing list)	http://www.15seconds.com
O'Reilly's Windows and VB sites detailing its resources and books	http://windows.oreilly.com http://vb.oreilly.com
Clarence Washington's repository for scripting solutions on the Internet	http://cwashington.netreach.net
Wrox publishes books on ADSI, ADO, VB, and WSH	http://www.wrox.com
Windows and .NET Magazine (formerly Windows 2000 Magazine) is published monthly, as is the Windows Scripting Solutions (formerly Win32 Scripting Journal), both of which provide a lot of good information on Active Directory and scripting	http://www.winnetmag.com http://www.win32scripting.com

18.6 Summary

Hopefully you now understand the basics of ADSI enough to be useful. It's a very robust API that allows you to interface to all aspects of both Active Directory and Windows NT, Windows 2000, and Windows Server 2003 servers. Even though the majority of this chapter covers Microsoft operating systems, the code does use the LDAP namespace and is portable to many other directory services. One of ADSI's biggest strengths is its ability to communicate with a variety of directory services using either LDAP or a provider-specific namespace.

In the next chapter, we will cover the IADs interface in more depth along with a discussion of the Property Cache. A chapter covering ADO will follow that, which should give you all the necessary tools to query and manipulate Active Directory.

Chapter 19. IADs and the Property Cache

Each object in a directory has a series of attributes, or properties, that uniquely define it. Although properties can vary from object to object, ADSI supports the manipulation of a core set of six properties common to all objects using the IADs interface. These properties are common to all objects because IADs is the most basic interface in ADSI.

19.1 The IADs Properties

The IADs properties are as follows:
Class

The object's schema class
GUID

The object's Globally Unique ID (GUID)
Name

The object's name
ADsPath

The ADsPath to the object in the current namespace
Parent

The ADsPath to the object's parent
Schema

The ADsPath to the object's schema class

Each of these properties has a corresponding property method in the IADs interface. You can use the property method, which has the same name as the property, to access that property's value. Example 19-1 contains code to display the six IADs properties for a user object.

Example 19-1. Using the explicit property methods to display the six IADs properties

```
 Dim objUser 'An ADSI User object
Dim str     'A text string

` User object using the WinNT namespace
Set objUser=GetObject("WinNT://MYCORP/Administrator,User")
str = "Name: " & objUser.Name & vbCrLf
str = str & "GUID: " & objUser.GUID & vbCrLf
str = str & "Class: " & objUser.Class & vbCrLf
str = str & "ADsPath: " & objUser.ADsPath & vbCrLf
str = str & "Parent: " & objUser.Parent & vbCrLf
str = str & "Schema: " & objUser.Schema & vbCrLf & vbCrLf
Set objUser = Nothing

` User object using the LDAP namespace
Set objUser=GetObject("LDAP://cn=Administrator,cn=Users,dc=mycorp,dc=com")
Str = str & "Name: " & objUser.Name & vbCrLf
Str = str & "GUID: " & objUser.GUID & vbCrLf
Str = str & "Class: " & objUser.Class & vbCrLf
Str = str & "ADsPath: " & objUser.ADsPath & vbCrLf
Str = str & "Parent: " & objUser.Parent & vbCrLf
str = str & "Schema: " & objUser.Schema & vbCrLf & vbCrLf

WScript.Echo str

Set objUser = Nothing
```

To begin, we declare two variables (i.e., str and objUser), invoke the GetObject method to create a reference to the user object, and assign it to objUser. We then set the str variable to the string "Name:" and apply the IADs::Name property method (i.e., objUser.Name) to retrieve the Name property's value (i.e., Administrator). The carriage-return line-feed constant (vbCrLf) specifies to move to the start of a new line. At this point, str represents the string "Name: Administrator."

In the next line, we use the IADs::GUID property method (objUser.GUID) to retrieve the GUID property's value

19.2 Manipulating the Property Cache

There will be times when you need to write a script that queries all the values that have been set in the underlying directory for a particular object. For example, suppose you're one of several systems administrators who work with your company's Active Directory implementation. You need to write a script that queries all the property values that the administrators have set for a particular user.

Discovering the set property values for an object can be a long, tedious job. Fortunately, ADSI provides a quick method. If someone has set a value for a property, it must be in that object's property cache. So all you need to do is walk through the property cache, displaying and optionally modifying each item as you go.

In this section, we'll describe the property cache mechanics and show you how to write scripts that use several ADSI methods and properties to add individual values, add a set of values, walk through the property cache, and write modifications to the cache and to the directory. Although these examples access the Lightweight Directory Access Protocol (LDAP) namespace, you can just as easily substitute the WinNT namespace in any of the scripts and run them against Windows NT servers.

Details of the property cache interfaces can be found at the MSDN Library (http://msdn.microsoft.com/library/) by clicking through the following links: Networking and Directory Services ⟶ Active Directory, ADSI, Directory Services ⟶ SDK Documentation ⟶ Directory Services ⟶ Active Directory Service Interfaces ⟶ Active Directory Service Interfaces Reference ⟶ ADSI Interfaces ⟶ Property Cache Interfaces.

19.2.1 Property Cache Mechanics

Every object has properties. When you perform an explicit IADs::GetInfo call (or an implicit IADs::GetInfo call using IADs::Get) on an object that you previously bound to, the OS loads all the properties for that specific object into that object's property cache. Consider the property cache a simple list of properties. The PropertyList object represents this list. You can use several IADsPropertyList methods to navigate through the list and access items. For example, you can navigate the list and access each item, every nth item, or one particular item based on its name.

Each item in the property list is a property entry represented by the PropertyEntry object. You use the IADsPropertyEntry interface to access property entries. A property entry can have one or more property values. To access values in a property entry, you use the IADsPropertyValue interface.

To summarize, use IADsPropertyList to navigate through and access property entries in the property list. When you want to manipulate a property, use IADsPropertyEntry. To access the values of that property entry, use IADsPropertyValue.

19.2.2 Adding Individual Values

To show you how to add an individual value, we'll expand on one of the examples from the previous section: the pager property of the User object. The pager property is an array of text strings representing multiple pager numbers.

Consider that any property represents data. Data can take several forms, including a string, an integer, or a Boolean value. In the cache, each property has two attributes: one attribute specifies the type of data the property represents, and the other attribute specifies the value of that data type. For example, each pager property has two attributes: a Unicode string (the type of data) and the pager number (the value of that Unicode string). The User object's lastLogon property, which specifies the time the user last logged on, has the two attributes, a LargeInteger (type of data) and a date/time stamp (the value of that LargeInteger).

The pager and lastLogon properties are instances of the PropertyValue object, so you manipulate them with the method and property methods of the IADsPropertyValue interface. For example, you use the IADsPropertyValue::ADsType property method to set the PropertyValue's type of data. Table 19-3 shows some of the corresponding constant names and values that you can set for the IADsPropertyValue::ADsType property.

Table 19-3. Constants for the IADsPropertyValue::ADsType property

19.3 Checking for Errors in VBScript

It is worthwhile to look at error handling in a little more detail now. Normally errors that occur in a script are termed fatal errors. This means that execution of the script terminates whenever an error occurs. When this happens, a dialog box opens and gives you the unique number and description of the error. While this is useful, sometimes you may like to set errors to be nonfatal, so that execution continues after the error. To do this, you include the following line in your code:

```
On Error Resume Next
```

Once you have done this, any line with an error is ignored. This can cause confusion, as can be seen from the following code. Note the missing P in LDAP:

```
On Error Resume Next

Set objGroup = GetObject("LDA://cn=Managers,ou=Sales,dc=mycorp,dc=com")

objGroup.GetInfo
WScript.Echo objGroup.Description
objGroup.Description = "My new group description goes here"
objGroup.GetInfo
WScript.Echo objGroup.Description
```

This script fails to execute any of the lines after the On Error Resume Next statement, as the first LDAP call into the objGroup variable failed. However, it will not terminate as usual with an error after the GetObject line, due to the On Error statement. To get around this, you should add a couple lines to do error checking. Example 19-9 is a good example of error checking in a different script.

Example 19-9. Error checking in VBScript

```
On Error Resume Next

'**********************************************************************
'Clear errors
'**********************************************************************
Err.Clear

'**********************************************************************
'Get a pointer to the Administrator account
'**********************************************************************
Set objUser = GetObject ("LDAP://cn=Administrator,cn=Users,dc=mycorp,dc=com")
If Hex(Err.Number)="&H80005000" Then
  WScript.Echo "Bad ADSI path!" & vbCrLf & "Err. Number: " _
    & vbTab & CStr(Hex(Err.Number)) & vbCrLf & "Err. Descr.: " _
    & vbTab & Err.Description
  WScript.Quit
End If

'**********************************************************************
'Explicitly call GetInfo for completeness
'**********************************************************************
objUser.GetInfo

'**********************************************************************
'Clear any previous errors
'**********************************************************************
Err.Clear

'**********************************************************************
'Try and get a pointer to the "moose" attribute of the user (which
'doesn't exist)
'**********************************************************************
x = objUser.Get("moose")

'**********************************************************************
'Check for property does not exist error
```

19.4 Summary

Over the last two chapters, we've covered the interfaces, methods, and property methods that allow you to use access and manipulate generic objects in Active Directory. These interfaces include:

- IADs
- IADsContainer (covered more fully later)
- IADsPropertyList
- IADsPropertyEntry
- IADsPropertyValue

We've also looked at how to supply credentials to authenticate with alternate credentials using the ADsOpenDSObject interface.

In the next chapter, we cover how to search Active Directory using a database query interface called ADO.

Chapter 20. Using ADO for Searching

Microsoft's ADO technology lets you conduct database searches and retrieve the results through a flexible interface called resultsets. ADO also lets you update information in a database directly or with stored procedures. Because Microsoft created an ADO database provider for ADSI (the ADSI OLE DB provider), you can also use ADO's database query technology to query Active Directory. However, the ADSI OLE DB provider is currently read-only, so many of the useful ADO methods for updating data aren't available yet. You can use ADO only for searching and retrieving objects. Despite the read-only limitation, using ADO is still a boon. It is significantly faster to search Active Directory using ADO than it is to use ADSI to bind to each object recursively down a branch. Even using IADsContainer::Filter is slow in comparison. So if you need to search Active Directory rapidly for attributes matching criteria, ADO is exactly what you should use. The ADO object model consists of nine objects (Command, Connection, Error, Field, Parameter, Property, Record, Recordset, and Streams) and four collection objects (Errors, Fields, Parameters, and Properties). However, some of these objects aren't useful if you're using the ADSI OLE DB provider, as they are more often used for accessing full-fledged database services. For example, the Parameter object lets you pass parameters to stored procedures, but this object is of little use because the ADSI provider doesn't support stored procedures.

The objects that are appropriate to ADSI in a read-only environment are the Command, Connection, Error, Field, Property, and Recordset objects. We use them to show you how to perform complex searches. For a full description of the ADO object model and the available functions, check out the following on the MSDN Library (http://msdn.microsoft.com/library/): Data Access ⟶ Microsoft Data Access Components (MDAC) ⟶ SDK Documentation ⟶ Microsoft ActiveX Data Objects (ADO).

If you wish to make use of the tools in this chapter in a VB project rather than a VBScript script, you need to include the Microsoft ActiveX Data Objects 2.x library from the Reference item on the Project menu of the Visual Basic Environment.

One point to note: ADO is written to work with all types of databases, so there are a numerous ways of doing exactly the same thing. We will attempt to cover examples of each different way as they crop up so that you will be able to choose the method that suits you best or that you are familiar with.

20.1 The First Search

The easiest way to explain basic searching using ADO is with an example. Here we'll build an ADO query to search and display the ADsPaths of all users in Active Directory. You can create a simple script to do this search in six steps.

20.1.1 Step 1—Define the Constants and Variables

For this script, you need to define one constant and three variables. The constant is adStateOpen, which we set to 1. If you're using VBScript, you use this constant later to determine whether you made a successful connection to the database. If you're using Visual Basic (VB), you don't have to include this constant because VB has already defined it. The two main variables are objConn (an ADO Connection object that lets you connect to the AD database) and objRS (an ADO Recordset object that holds the retrieved resultset). The third variable holds the output of the resultset, as shown in the following example:

```
Option Explicit

Const adStateOpen = 1

Dim objConn    'ADO Connection object
Dim objRS      'ADO Recordset object
Dim strOutput  'The output of the search
```

The Option Explicit statement at the beginning of the script is optional, but we recommend that you include it. This statement forces the script to declare variables, so you can quickly spot errors.

20.1.2 Step 2—Establish an ADO Database Connection

To perform an ADO query, you need to establish an ADO connection, which is completely separate from any ADSI connections you may have opened with IADsOpenDSObject::OpenDSObject. Before you can establish this connection, you must create an ADO Connection object to use. This object can be created the same way you create a file system object: use the CreateObject method, with "ADODB.Connection" as a parameter. You use the ADODB prefix to create all ADO objects, and Connection is the top-level object in the ADO object model:

```
Set objConn = CreateObject("ADODB.Connection")
```

Just as you use different programmatic identifiers (ProgIDs) (e.g., WinNT:, LDAP:) to tell ADSI which directory to access, you use different OLE DB providers to tell ADO which query syntax to use. An OLE DB provider implements OLE DB interfaces so that different applications can use the same uniform process to access data. The ADSI OLE DB connector supports two forms of syntax: the SQL dialect and the LDAP dialect. Although you can use the SQL dialect to query the ADSI namespace, most scriptwriters use the LDAP dialect because Microsoft defined it specifically for ADO queries to directory services. However, the default for the Connection object's read/write property, objConn.Provider, is MSDASQL, which specifies the use of SQL syntax. Because you want to use the ADSI provider, you need to set objConn.Provider to "ADsDSOObject", which specifies the use of the LDAP syntax. By setting this specific provider, you force the script to use not only a specific syntax but also a specific set of arguments in the calls to the Connection object's methods.

```
objConn.Provider = "ADSDSOObject"
```

20.1.3 Step 3—Open the ADO Connection

You can open a connection to the directory by calling the Connection::Open method. When describing the methods and property methods of COM interfaces in text, the established notation is to use a double colon (::) separator. For example, Connection::Open specifies the Open method of the Connection object, as shown in the following example:

```
objConn.Open _
 "", "CN=Administrator,CN=Users,dc=mycorp,dc=com", ""
```

As the code shows, the Open method takes three parameters. The first parameter is the Connection::ConnectionString parameter, which contains information that the script needs to establish a connection to the data source. In this case, it is blank. The second parameter contains the user DN to bind with, and the third is the user's password.

20.2 Other Ways of Connecting and Retrieving Results

As mentioned earlier, there are a number of ways of authenticating an ADO connection to Active Directory. The simplest is the way outlined earlier using the Connection::Provider set with the username and password as second and third arguments:

```
 Set objConn = CreateObject("ADODB.Connection")
objConn.Provider = "ADSDSOObject"
objConn.Open "", _
             "CN=Administrator,CN=Users,dc=mycorp,dc=com", _
             "mypass"
```

Because ADO is designed for databases, it is often necessary to specify a number of other requirements when opening a connection. These include a different provider, a different server, or a specific database. All of these items can be set prior to opening the connection. However, none of these make a difference to the AD provider. If you wish to open a connection by setting these values in the ConnectionString property, then do so as shown in the following code:

```
 Set objConn = CreateObject("ADODB.Connection")
objConn.Provider = "ADSDSOObject"objConn.ConnectionString = _
  "DSN=;UID=CN=Administrator,CN=Users,dc=mycorp,dc=com;PWD=mypass"
objConn.Open
```

Semicolons separate the arguments, with the expected DataSourceName (DSN) specified as empty at the start of the string.

One important point: do not authenticate using both methods with the same connection—use one or the other. The following code uses both methods to illustrate what not to do:

```
 Set objConn = CreateObject("ADODB.Connection")
objConn.Provider = "ADSDSOObject"
objConn.Open _
  "DSN=;UID=CN=Administrator,CN=Users,dc=mycorp,dc=com;PWD=mypass", _
  "CN=Administrator,CN=Users,dc=mycorp,dc=com", "mypass"
```

This is a slightly different version, but still wrong:

```
 Set objConn = CreateObject("ADODB.Connection")
objConn.Provider = "ADSDSOObject"
objConn.ConnectionString = _
  "DSN=;UID=CN=Administrator,CN=Users,dc=mycorp,dc=com;PWD=mypass"
objConn.Open "", "CN=Administrator,CN=Users,dc=mycorp,dc=com", _
  "mypass"
```

20.2.1 Searching With SQL

You can retrieve resultsets in a variety of ways and get exactly the same values. We will now discuss how to use the Command object and the Recordset::Open method, using SQL-formatted queries to retrieve resultsets. SQL is a powerful query language that is the de facto standard to query database tables. We do not propose to go through the details of SQL here, but we will cover some examples for those who may already be familiar with SQL and would find using it to be a more comfortable way of querying Active Directory than using LDAP search filters.

20.2.1.1 Using the Connection::Execute method

You can pass a SQL select statement to a connection using the Execute method as we've done previously with LDAP-based queries:

```
 Set objConn = CreateObject("ADODB.Connection")
objConn.Provider = "ADSDSOObject"
objConn.Open "", "CN=Administrator,CN=Users,dc=mycorp,dc=com", ""

Set objRS = objConn.Execute "Select Name, ADsPath" _
  & " FROM 'LDAP://dc=mycorp,dc=com' where objectclass = 'user'"
```

20.2.1.2 Using the Recordset::Open method

20.3 Understanding Search Filters

When you use the LDAP dialect with the ADSI OLE DB provider to conduct a search, you must use an LDAP search filter to specify your search criteria. In a simple case, (objectclass=user) would be used to select every object with the user objectclass under the search base. You can in fact use a filter to match the presence of a value (or not) for any attribute of an object. This enables you to create powerful searches with complex criteria. For example, you can search for any group object that has a certain user as a member and that has a description matching a certain substring.

Filters must follow the format specified in RFC 2254. You can download RFC 2254 from http://www.ietf.org/rfc/rfc2254.txt.

Although filters let you conduct powerful searches, working with them can seem complex because of the format used, known as prefix notation. To make it easier to understand, we have divided the discussion of filters into two parts: items within a filter and items connecting filters.

20.3.1 Items Within a Filter

Within a filter, you can have three types of items:
Operators

A filter can include one of three operators. The equal-to (=) operator checks for exact equivalence. An example is (name=janet). The greater-than-or-equal-to (>=) and less-than-or-equal-to (<=) operators check for compliance with a range. Examples are (size>=5) and (size<=20).
Attributes

You can include attributes in filters when you want to determine whether an attribute exists. You simply specify the attribute, followed by the = operator and an asterisk (*). For example, the (mooseHerderProperty=*) filter searches for objects that have the mooseHerderProperty attribute populated.
Substrings

You can include substrings in filters when you want to search for objects with specific strings. Test for substrings by placing the attribute type (e.g., cn for common name, sn for surname) to the left of the = operator and the substring you're searching for to the right. Use the * character to specify where that substring occurs in the string. The (cn=Keith*) filter searches for common name (CN) attributes that begin with the substring "Keith"; the (cn=*Cooper) filter searches for CN strings that end with the substring "Cooper". Depending on the search, the latter form of substring searches can take a long time to return. Under Windows Server 2003, the substring searches perform much better than previously.

You can place several substrings together by using an asterisk character several times. For example, the (cn=Kei*Coo*) filter searches for two substrings in the string: the first substring begins with "Kei", followed by the second substring that begins with "Coo". Similarly, the (cn=*ith*per) filter searches for strings that have two substrings: the first substring ends in "ith" followed by the second substring that ends in "per".

The resultset of a substring search might contain objects that you don't want. For example, if you use the filter (cn=Kei*Coo*) to search for the object representing "Keith Cooper", your resultset might contain two objects: one representing "Keith Cooper" and another representing "Keith Coolidge". To address that issue, you can connect multiple filter strings together to refine your search even more.

20.3.2 Connecting Filters

Compound filters can be created by using the ampersand (&), the vertical bar (|), and the exclamation mark (!). Let's start by creating a filter to find all groups whose common name begins with the letter a. The following is the filter for this search:

20.4 Optimizing Searches

Whether you are searching Active Directory using filters or with SQL, there are some important guidelines to follow that can help reduce load on the domain controllers, increase performance of your scripts and applications, and reduce the amount of traffic generated on the network. It is also important to socialize these concepts with others as much as possible. It takes only a couple of badly written search filters in a heavily used application to severely impact the performance of your domain controllers!

20.4.1 Efficient Searching

Understanding how to write efficient search criteria is the first important step to optimizing searches. By understanding a few key points, you can greatly improve the performance of your searches. It is also important to reuse data retrieved from searches or connections to Active Directory as much as possible. The following list describes several key points to remember about searching:

- Use at least one indexed attribute per search. Certain attributes are marked as "indexed" in Active Directory, which allows for fast pattern matching. They are typically single-valued and unique, which means searches using indexed attributes can determine which objects match them very quickly. If you don't use indexed attributes, the database equivalent of a full table scan must be done to determine the matches.
- Use a combination of objectclass and objectcategory in every search. While most of the queries used so far in this chapter have used only objectclass, you should make it a practice always to use a combination of objectclass and objectcategory. The problem with using only objectclass is that it is not indexed because it is multivalued and not unique, while objectcategory is single-valued and indexed. See the next section Section 20.4.2 for more information.
- Try to limit the use of trailing (name=*llen) or middle match (name=*lle*) searches. Unlike other directories, Active Directory is not optimized to handle these types of searches, and they should be avoided if possible. In some cases these types of searches can take upwards of 10-15 seconds to complete under Windows 2000!
- Use the appropriate search scope. Avoid using subtree searches unless you truly want to search more than one level down. If you only want to search directly below the search base, use the OneLevel scope.
- Use paged searching for queries that can potentially return thousands of entries. Most subtree searches should have paging enabled unless you are positive the search will not return more than 1,000 entries or do not want it to return more than 1,000 entries.
- Reuse ADO Connection and Command objects as much as possible. ADO Connection and Command objects can be used for multiple searches so there is no need to create additional ones.

20.4.2 Objectclass Versus Objectcategory

It is very important to understand the differences between objectclass and objectcategory and how they should be used during searches. Objectclass is a multi-valued attribute that contains the objectclass hierarchy for an instantiated object. For example, a user object has the following values as part of its objectclass attribute:

- top
-

20.5 Advanced Search Function—SearchAD

We will now take many of the concepts from this chapter and apply them in a useful example called SearchAD. SearchAD can be included in any VBScript and used immediately as is.

SearchAD takes five parameters and returns a Boolean indicating whether it succeeded or failed in the search. You should recognize most of these parameters.

- The base ADsPath to start the search from
- A valid ADO criteria string
- The depth that you wish to search, represented by one of the exact strings Base, OneLevel, or SubTree
- The comma-separated list of attributes that is to be returned
- A variable that will hold the returned results of the search in an array

The last parameter does not have any values when passed in, but if SearchAD is successful, the array contains the resultset.

Here is an example use of SearchAD:

```
bolIsSuccess = SearchAD("LDAP://ou=Finance,dc=mycorp,dc=com", _
  "(cn=a*)", "Base", "cn,description", arrSearchResults)
```

You can also use it as part of an If...Then condition:

```
If SearchAD("LDAP://dc=mycorp,dc=com", "(description=moose)", "SubTree", _
  "ADsPath,cn,description", arrSearchResults) Then
  'success code using arrSearchResults
Else
  'failure code
End If
```

The array that is returned is a two-dimensional array of attributes that match the criteria. If there were 12 results returned for the preceding query, this is how you access the results:

```
arrSearchResults(0,0) 'ADsPath of first result
arrSearchResults(0,1) 'CN of first result
arrSearchResults(0,2) 'Description of first result
arrSearchResults(1,0) 'ADsPath of second result
arrSearchResults(1,1) 'CN of second result
arrSearchResults(1,2) 'Description of second result
arrSearchResults(2,0) 'ADsPath of third result
arrSearchResults(2,1) 'CN of third result
arrSearchResults(2,2) 'Description of third result
arrSearchResults(3,0) 'ADsPath of fourth result
arrSearchResults(3,1) 'CN of fourth result
arrSearchResults(3,2) 'Description of fourth result
.
.
.
arrSearchResults(11,0) 'ADsPath of 11th result
arrSearchResults(11,1) 'CN of 11th result
arrSearchResults(11,2) 'Description of 11th result
```

You can loop through these values in your own code using VBScript's built-in function UBound to find the maximum

20.6 Summary

In this chapter, we reviewed the basics of ADO, which provides a robust search interface for Active Directory. While originally intended for databases, ADO was adapted to Active Directory to allow queries based on LDAP search filters or SQL. Several techniques for optimizing searches in Active Directory were reviewed, including a discussion of using objectclass versus objectcategory. We ended the chapter by covering a fully functional SearchAD procedure that can be used as is in any VBScript to easily search Active Directory based on specified criteria. SearchAD hides all the underlying ADO logic, including connection setup, query execution, and recordset manipulation.

After providing a good background for ADSI and ADO in Chapter 18 through Chapter 20, we are now ready to move to more practical applications. The next several chapters show some of the capabilities these interfaces provide and a lot of sample code to get you started.

Chapter 21. Users and Groups

In this chapter, we will show you how to automate the creation and manipulation of user and group accounts. Although tools to create user and group accounts already exist (e.g., the Resource Kit's Addusers utility), ADSI's versatility lets you quickly write a script that creates 1,000 fully featured user or group accounts based on whatever business logic you require. You can also create command-line utilities or web-based interfaces using the techniques shown in this chapter to perform such functions as unlocking locked-out user accounts or adding users to groups.

21.1 Creating a Simple User Account

You can quickly create a user account with minimal attributes with ADSI. The following code shows how to create a user in an NT domain, a local computer, and an Active Directory domain.

```
 Option Explicit
Dim objDomain, objUser
'Creating a user in a Windows NT domain

Set objDomain = GetObject("WinNT://MYDOMAIN")
Set objUser = objDomain.Create("user","vlaunders")
objUser.SetInfo

'Creating a local user on a computer or member server
'Valid for Windows NT/2000/2003
Set objComputer = GetObject("WinNT://MYCOMPUTER,Computer")
Set objUser = objComputer.Create("user","vlaunders")
objUser.SetInfo

'Creating a user in Active Directory
Set objDomain = GetObject("LDAP://cn=Users,dc=mycorp,dc=com")
Set objUser = objDomain.Create("user","cn=vlaunders")
objUser.Put "sAMAccountName", "vlaunders"
objUser.Put "userPrincipalName", "vlaunders@mycorp.com"
objUser.SetInfo
```

The code is composed of three sections. The first two sections use the WinNT provider to create a user account in an NT 4.0 domain, and in a computer that could be a member server or part of a workgroup. The third section uses the LDAP provider to create a user account in an Active Directory domain.

When you create users in an Active Directory domain, you need to be aware of two important User object attributes: sAMAccountName and userPrincipalName. The User object has several mandatory attributes. The system sets many of these mandatory attributes, except for one, sAMAccountName, which allows Active Directory-based clients to interact with older clients and NT domains. You must set the sAMAccountName attribute before you call IADs::SetInfo or the creation will fail. The userPrincipalName attribute isn't mandatory, but it is recommend so users can log on using an email-style address as defined in RFC 822 (http://www.ietf.org/rfc/rfc822.txt).

21.2 Creating a Full-Featured User Account

Creating user accounts as we've done previously is fine for an introduction, but typically you'll need to set many more attributes to make them usable in your environment. The approaches you use to create fully featured users in the NT and Active Directory environments differ slightly; Active Directory offers considerably more properties than NT, such as the office and home addresses of users, as well as lists of email addresses and pager, fax, and phone numbers.

You can manipulate User objects with a special interface called IADsUser. IADsUser's methods and property methods let you directly set many of the User object's property values. Table 21-1 through Table 21-3 contain the methods, read-write property methods, and read-only property methods, respectively, for the IADsUser interface. The corresponding Active Directory attribute is included in parentheses for the property methods that can be set with the LDAP provider.

Table 21-1. IADsUser methods

Method	**Description**
IADsUser::ChangePassword	Changes the existing password.
IADsUser::SetPassword	Sets a new password without needing the old one.
IADsUser::Groups	Gets a list of groups of which the user is a member. You can use the IADsMembers interface to iterate through the list.

Table 21-2. IADsUser read-write property methods

Property method	**Available with WinNT or LDAP?**
IADsUser::AccountDisabled	WinNT, LDAP (userAccountControl mask)
IADsUser::AccountExpirationDate	WinNT, LDAP (accountExpires)
IADsUser::Department	LDAP (department)
IADsUser::Description	WinNT, LDAP (description)
IADsUser::Division	LDAP (division)
IADsUser::EmailAddress	LDAP (mail)
IADsUser::EmployeeID	LDAP (employeeID)
IADsUser::FaxNumber	LDAP (facsimileTelephoneNumber)
IADsUser::FirstName	LDAP (givenName)
IADsUser::FullName	WinNT, LDAP (displayName)

21.3 Creating Many User Accounts

User-specific scripts work well if you have to create only a few user accounts. If you need to create many user accounts at one time, or if you create new accounts often, using a script with an input file is more efficient. The input file includes the user data so that you can use the script to create any user account. For example, the output shown below represents the users-to-create.txt input file that provides the user data for the universal script in Example 21-3. Although this input file includes only four data sets, you can include as many data sets as you want. You include a data set for each user account that you want to create.

```
 vlaunders:12/09/01:The description:Victoria Launders:onebanana
aglowenorris:08/07/00:Another user:Alistair Lowe-Norris:twobanana
kbemowski:03/03/03:A third user:Karen Bemowski:threebanana
jkellett:08/09/99:A fourth user:Jenneth Kellett:four
```

As the output shows, each data set goes on a separate line. A data set can contain as many values as you want. The data sets in the users-to-create.txt file have five values: username, expiration date, description, full name, and password. You use colons to separate the values.[1]

[1] While comma-separate-value (CSV) files are the norm for this sort of thing, the comma is more often used in properties that will be added for users, so I use the colon here instead.

Example 21-3. Creating many user accounts using a script with an input file

```
 Option Explicit

Const ForReading = 1

Dim objDomain, objUser, fso, tsInputFile, strLine, arrInput
Dim fldUserHomedir, wshShell

Set objDomain = GetObject("LDAP://cn=Users,dc=mycorp,dc=com")
Set fso = CreateObject("Scripting.FileSystemObject")

'*********************************************************************
'Open the text file as a text stream for reading.
'Don't create a file if users-to-create.txt doesn't exist
'*********************************************************************
Set tsInputFile = fso.OpenTextFile("c:\users-to-create.txt", ForReading, False)

'*********************************************************************
'Execute the lines inside the loop, even though you're not at the end
'of the file
'*********************************************************************
While Not tsInputFile.AtEndOfStream

  '*******************************************************************
  'Read a line, and use the Split function to split the data set into
  'its separate parts
  '*******************************************************************
  strLine = tsInputFile.ReadLine
  arrInput = Split(strLine, ":")

  Set objUser = objDomain.Create("user","cn=" & arrInput(0))
  objUser.Put "sAMAccountName", & arrInput(0)
  objUser.Put "userPrincipalName", arrInput(0) & "@mycorp.com"

  '*******************************************************************
  'Write the newly created object out from the property cache
  'Read all the properties for the object, including
  'the ones set by the system on creation
  '*******************************************************************
  objUser.SetInfo
  objUser.GetInfo

  '*******************************************************************
```

21.4 Modifying Many User Accounts

Once you have created the user accounts in a domain, you will more than likely need to modify them at some point. The modifications may consist only of changing individual properties of a user, such as the description or name fields. In these cases, you can perform the change manually or write a command-line script as shown in the next section. In some situations, you will need to make a large number of changes to your user accounts, as would be the case if you changed the name of your login script and wanted to point all users at the new script.

For Windows NT and even Active Directory domains, you can use the IADsContainer::Filter method to iterate through all the objects of a particular type. Thus, changing all users' login script is a pretty easy to do:

```
 Option Explicit
On Error Resume Next
Dim objDomain, objUser
Set objDomain = GetObject("WinNT://MYCORP")
objDomain.Filter = Array("User")
'*********************************************************************
' Iterate over each user and set the LoginScript
' Print an error if one occurs
'*********************************************************************
for each objUser in objDomain
   objUser.LoginScript = "login-new.vbs"
   objUser.SetInfo

   if Err.Number <> 0 Then
      Wscript.Echo objUser.Name & " error occurred"
      Err.Clear
   Else
      Wscript.Echo objUser.Name & " modified"
   End if
next
```

While the previous code is straightforward, it is also limiting. The only filter option you have is object type, such as all users, and no additional criteria are allowed. That is why in most cases with Active Directory domains, you will want to use ADO to find objects, as explained in Chapter 20. So for our next example, let's say that we want to change the login script for all users in the domain that have a department attribute equal to "Sales". Example 21-4 shows how this can be done using ADO.

Example 21-4. Modifying the login script for all users in Sales

```
 Option Explicit
On Error Resume Next
Dim objConn, objComm, objRS, objUser
Dim strBase, strFilter, strAttrs, strScope
'*********************************************************************
'Set the ADO search criteria
'*********************************************************************
strBase   = "<LDAP://dc=mycorp,dc=com>;"
strFilter = "(&(objectclass=user)(objectcategory=Person)(department=Sales));"
strAttrs  = "ADsPath;"
strScope  = "Subtree"

set objConn = CreateObject("ADODB.Connection")
objConn.Provider = "ADsDSOObject"
objConn.Open
'*********************************************************************
'Need to enable Paging in case there are more than 1000 objects returned
'*********************************************************************
Set objComm = CreateObject("ADODB.Command")
Set objComm.ActiveConnection = objConn
objComm.CommandText = strBase & strFilter & strAttrs & strScope
objComm.Properties("Page Size") = 1000
Set objRS = objComm.Execute(  )
While not objRS.EOF
  Set objUser = GetObject( objRS.Fields.Item("ADsPath").Value )
```

21.5 Account Unlocker Utility

Imagine that you need a utility that quickly enables and unlocks an NT or Active Directory user account. The account was locked because the password was entered incorrectly too many times in succession or because the account exceeded its expiration date. Writing a user-specific script is inefficient if you have many users. Using an input file to pass in the needed user data to a script also is inefficient. You'd have to create the input file just before running the script, because you can't predict whose account you need to unlock. The best approach is to use command-line arguments to pass in the user data as you need it.

Example 21-5 and Example 21-6 use this approach to enable and unlock NT and Active Directory user accounts, respectively. If you have a mixed NT and Active Directory network, you can even combine these two utilities into one script.

Example 21-5 implements the unlocker with the WinNT provider.

Example 21-5. Account unlocker utility for Windows NT

```
'*********************************************************************
'How to unlock and enable a Windows NT user via arguments to this script
'
'Parameters should be <domain> <username>
'*********************************************************************
Option Explicit

Dim wshArgs, objUser, strOutput

On Error Resume Next

'*********************************************************************
'Get the arguments
'*********************************************************************
Set wshArgs = Wscript.Arguments

'*********************************************************************
'If no arguments passed in, then quit
'*********************************************************************
If wshArgs.Count = 0 Then
  WScript.Echo "ERROR: No arguments passed in." & vbCrLf & vbCrLf _
    & "Please use NTUNLOCK <domain> <username>" & vbCrLf & vbCrLf
  WScript.Quit
End If

'*********************************************************************
'Error checking of the arguments could go here if we were bothered
'*********************************************************************

'*********************************************************************
'Attempt to bind to the user
'*********************************************************************
Set objUser = GetObject("WinNT://" & wshArgs(0) & "/" & wshArgs(1) & ",user")
If Err Then
  Wscript.Echo "Error: Could not bind to the following user: " & vbCrLf _
    & vbCrLf & "WinNT://" & wshArgs(0) & "/" & wshArgs(1) & vbCrLf & vbCrLf
  WScript.Quit
Else
  strOutput = "Connected to user WinNT://" & wshArgs(0) & "/" & wshArgs(1) _
    & vbCrLf
End If

'*********************************************************************
'Attempt to enable the user (but don't quit if you fail)
'*********************************************************************
Err.Clear
objUser.AccountDisabled = False
```

21.6 Creating a Group

Now we will move on to creating groups. Creating a group is very similar to creating a user. You use the same IADsContainer::Create method:

```
 Set objGroup = objSalesOU.Create("group", "cn=Managers")
objGroup.Put "sAMAccountName", "Managers"
objGroup.SetInfo
```

This code assumes we already have a pointer to an OU in the objSalesOU variable. The IADs::Put method is used to set the sAMAccountName, a mandatory attribute with no default value, just like with users.

The IADsGroup interface that operates on group objects supports four methods and one property that is specific to the group object, as listed in Table 21-4.

Table 21-4. The IADsGroup interface

IADsGroup methods and properties	Action
Add	Adds users to the group as members
Remove	Removes user members from the group
IsMember	Tests to see if a user is a member of a group
Members	Returns a list of all the members of the group
Description	Returns the text describing the group

In Example 21-7, we show how to create a group with both the WinNT and LDAP providers.

Example 21-7. Creating a group with both the WinNT and LDAP providers

```
 Option Explicit

Dim objDomain, objGroup

'Creating a group in a Windows NT domain
Set objDomain = GetObject("WinNT://MYDOMAIN")
Set objGroup = objDomain.Create("group","My Group")
ObjGroup.SetInfo

'Creating a local group on a computer or member server

'Valid for Windows NT, Windows 2000 and Windows Server 2003
Set objComputer = GetObject("WinNT://MYCOMPUTER,Computer")
Set objGroup = objComputer.Create("group","My Group")
ObjGroup.SetInfo

'Creating a group in Active Directory
Set objDomain = GetObject("LDAP://cn=Users,dc=mycorp,dc=com")
Set objGroup = objDomain.Create("user","cn=My Group")
ObjGroup.Put "sAMAccountName", "MyGroup"
ObjGroup.SetInfo
```

21.7 Adding Members to a Group

Adding objects as members of a group can be done with IADsGroup::Add, a simple method that takes the DN of the object to be added:

```
objGroup.Add("LDAP://cn=Sue Peace,cn=Users,dc=mycorp,dc=com")
objGroup.Add("LDAP://cn=Keith Cooper,cn=Users,dc=mycorp,dc=com")
```

Groups can contain virtually any other type of object as a member, including users, computers, and other groups.

21.7.1 Adding Many USER Groups to DRUP Groups

In Section 11.5.5, we described the need to add many user groups as members of several permission groups. Example 21-8 contains the code necessary to implement this functionality. It scans for all groups prefixed with USER_ and DRUP_. It then adds all the USER groups to each DRUP group, except for the group where the suffix matches. In other words, all USER_ groups except USER_Finance are added to DRUP_Finance. This was why the names were set up this way.

These searches make use of the ADO search function called SearchAD from Chapter 20.

Example 21-8. Adding many user groups as members of several permission groups

```
'**************************************************************************
'Search the entire AD for all groups starting USER_ and return the cn
'and AdsPath variables in the following structure
'
'  arrUSERGroup(0,index) = cn attributes
'  arrUSERGroup(1,index) = ADsPath attribute
'
'where index goes from 0 to (the maximum number of results returned -1)
'**************************************************************************
If SearchAD( _
  "LDAP://dc=mycorp,dc=com", "(&(objectClass=group)(cn=USER_*))", _
  "SubTree", "cn,ADsPath", arrUSERGroup) Then

  '************************************************************************
  'As above but for DRUP_ groups
  '************************************************************************
  If SearchAD( _
    "LDAP://dc=mycorp,dc=com", "(&(objectClass=group)(cn=DRUP_*))", _
    "SubTree", "cn,ADsPath", arrDRUPGroup) Then

    '**********************************************************************
    'Set up an index to allow us to iterate through the USER_ groups. The
    'Ubound function here counts the maximum number of elements in the
    'array's second dimension of values (the first dimension has only two
    'values, "cn" and "ADsPath")
    '**********************************************************************
    For intUSERGroupIndex = 0 To Ubound(arrUSERGroups,2)
      '********************************************************************
      'As above but for DRUP_ groups
      '********************************************************************
      For intDRUPGroupIndex = 0 To Ubound(arrDRUPGroups,2)
        '******************************************************************
        'Extract the portion of the name that corresponds to all letters after
        'the "cn=USER_" or "cn=DRUP_" parts (i.e., eight  letters)
        '******************************************************************
        txtUSERGroupSuffixName = Right(arrUSERGroup(0,intUSERGroupIndex), _
          Len(arrUSERGroup(0,intUSERGroupIndex))-8)
        txtDRUPGroupSuffixName = Right(arrDRUPGroup(0,intDRUPGroupIndex), _
          Len(arrDRUPGroup(0,intDRUPGroupIndex))-8)
        '******************************************************************
```

21.8 Evaluating Group Membership

The IADsGroup::IsMember method takes one argument, the DN of the object to check, just as Add and Remove do. It returns a Boolean, i.e., true or false. That allows you to use it in an If . . . Then statement like this:

```
 Set objGroup = GetObject("LDAP://cn=Managers,ou=Sales," _
  & "dc=mycorp,dc=com")
If objGroup.IsMember("LDAP://cn=Vicky Launders,ou=Sales," _
  & "dc=mycorp,dc=com") Then
    WScript.Echo "Is a Member!"
Else
    WScript.Echo "Is NOT a Member!"
End If
```

This should seem fairly straightforward after the examples we've already gone through. Two of the lines in the previous code snippet are too long to fit on the page, so the VBScript underscore (_) character was used again to tell VBScript that it should treat the current line as continuous with the next line. However, when you use the underscore to separate long strings, you must enclose both strings in quotation marks and then use the ampersand character (&) to concatenate two strings together.

To get a list of members in a group, the IADsGroup::Members method can be used. The IADsGroup::Members function is different from the other IADsGroup methods we have shown so far, since it returns a pointer to an IADsMembers object. Table 21-5 shows the two methods IADsMembers support.

Table 21-5. The IADsMembers interface

IADsMembers methods	**Action**
Count	The number of items in the container. If there is a filter set, only the number of items that match the filter are returned.
Filter	A filter, consisting of an array of object class strings, which can restrict the number of objects returned during enumeration of the container.

There are a number of ways of enumerating the members of a group. The For Each . . . In . . . Next loop is the most common. This is how it works:

```
 Set objGroup = GetObject("LDAP://cn=Managers,ou=Sales," _
  & "dc=mycorp,dc=com")
WScript.Echo "Number of members of the group: " & objGroup.Members.Count
For Each objMember In objGroup.Members
  WScript.Echo objMember.Name
Next
```

This script displays the number of members and then prints each member's name. As the For loop executes, objMember ends up holding an IADs object representing each member of the group.

Another useful feature of IADsMembers is the Filter method. It can be used to filter certain object classes during enumeration just like you can with containers. To view only the members of a group that are users, you would modify the previous example to do the following:

```
 objMembers = objGroup.Members
objMembers.Filter = Array("User")
For Each objMember In objMembers
  WScript.Echo objMember.Name
Next
```

21.9 Summary

In this chapter, we looked at how to create and manipulate properties of user and group objects in Active Directory and the Windows NT SAM. We used this knowledge to show how to write a script to create thousands of users easily from a set of data in a file or from a database. We then showed how to create simple tools, such as an account unlocker, that you can use in your day-to-day management of Active Directory. Next we showed how to create groups and modify group members. Finally, we reviewed how to determine group membership and iterate through all the members of a group.

Chapter 22. Manipulating Persistent and Dynamic Objects

ADSI can be used for much more than just user, group, or generic directory manipulation. ADSI provides many interfaces that you can use to manipulate persistent and dynamic objects for a computer. Persistent objects are permanent parts of a directory or computer, such as shares, services, users, and groups. Dynamic objects aren't permanent but instead are things such as sessions (i.e., connections to a machine) and print jobs that a user initiates. In other words, ADSI lets you do the following:

- Dynamically start, stop, and manage services and manipulate the permanent attributes of those services
- Dynamically manipulate shares, creating and deleting them as required
- Dynamically manipulate computers' open resources and users' active sessions and manipulate the permanent objects representing those computers and users
- Dynamically manipulate print jobs and manipulate the permanent queues

Many of you may already be familiar with the Windows Management Instrumentation (WMI) interface, which overlaps with several of these functions. Depending on your preference, you can use ADSI or WMI for many of these tasks. We describe WMI in more detail in Chapter 26.

22.1 The Interface Methods and Properties

Rather than describe the various methods and properties as we've done with the earlier interfaces, we'll concentrate on how to use those methods and properties in scripts. You can find complete descriptions of the interface methods and properties we cover in the MSDN Library or Platform SDK. To access the descriptions on the MSDN web site (http://msdn.microsoft.com/library/), navigate to Networking and Directory Services ⟶ Active Directory, ADSI and Directory Services ⟶ SDK Documentation ⟶ Directory Services ⟶ Active Directory Service Interfaces ⟶ Active Directory Service Interfaces Reference ⟶ ADSI Interfaces. From this point, you can navigate to:

Core Interfaces

IADs, IADsContainer, IADsNamespaces, and IADsOpenDSObject

Persistent Object Interfaces

IADsCollection, IADsFileShare, IADsService, IADsPrintJob, and IADsPrintQueue

Dynamic Object Interfaces

IADsServiceOperations, IADsComputerOperations, IADsFileServiceOperations, IADsResource, IADsSession, IADsPrintJobOperations, and IADsPrintQueueOperations

Utility Interfaces

IADsADSystemInfo, IADsDeleteOps, IADsNameTranslate, IADsObjectOptions, IADsPathname, and IADsWinNTSystemInfo

The ADSI documentation, however, leaves out three important quirks of the IADsSession and IADsResource interfaces. First, the WinNT provider doesn't currently support the IADsSession::UserPath, IADsSession::ComputerPath, and IADsResource::UserPath property methods. Second, although the documentation states that the IADsSession::ConnectTime and IADsSession::IdleTime property methods return results in minutes, they actually return results in seconds. Finally, the IADsSession::Computer property method returns NetBIOS names for Windows NT and Windows 9x clients but returns TCP/IP addresses for Windows 2000 and later clients.

22.2 Creating and Manipulating Shares with ADSI

The following code shows how easily you can create shares with ADSI:

```
Dim objComputer, objFileShare

Set objComputer = GetObject("WinNT://mydomainorworkgroup/mycomputer/LanmanServer")

Set objFileShare = objComputer.Create("FileShare", "MyNewShare")
objFileShare.Path = "c:\mydirectory"
objFileShare.Description = "My new Share"
objFileShare.MaxUserCount = 8
objFileShare.SetInfo
```

After we declare the objComputer and objFileShare variables, we bind to the LanmanServer object on the computer on which we want to create the shares. LanmanServer is the object name of the server service that runs on all Windows NT and later computers. We bind to this object because NT's predecessor was LAN Manager and is still present to a large extent in the Windows OS.

Next, we use the IADsContainer::Create method to create an object of class FileShare and apply the IADsFileShare property methods to set the path, description, and maximum number of users. On an NT, Windows 2000, or Windows Server 2003 server, you can grant all users access to a share or limit access to as many users as you want. On a workstation, you can grant all users access to a share or limit access to between 1 and 10 users at a time. The latter restriction is due to the 10-connection limit that the OS imposes. The values that the IADsFileShare::MaxUserCount method accepts are -1 (which grants all users access), any numerical value between 1 and 10 on workstations, and, within reason, any numerical value on the server family of OSs.

Finally, we end the script with IADs::SetInfo, which writes the information from the property cache to the directory.

Enumerating existing shares is just as easy as creating them. The next piece of code shows how to enumerate normal shares.[1]

[1] Hidden shares aren't shown due to their very nature.

```
Dim objService, objFileShare, strOutput

strOutput = ""

Set objService = GetObject("WinNT://workgroup/vicky/LanmanServer")

For Each objFileShare In objService
  strOutput = strOutput & "Name of share : " & objFileShare.Name & vbCrLf
  strOutput = strOutput & "Path to share : " & objFileShare.Path & vbCrLf
  strOutput = strOutput & "Description : " & objFileShare.Description & vbCrLf

  If objFileShare.MaxUserCount = -1 Then
    strOutput = strOutput & "Max users : No limit" & vbCrLf
  Else
    strOutput = strOutput & "Max users : " & objFileShare.MaxUserCount & vbCrLf
  End If

  strOutput = strOutput & "Host Computer : " _
    & objFileShare.HostComputer & vbCrLf & vbCrLf
Next

WScript.Echo strOutput
```

This code is similar to that in the previous script for creating a share. This is a sample of the output:

```
Name of share : NETLOGON
Path to share : C:\WINNT35\system32\Repl\Import\Scripts
Description   : Logon server share
Max users     : No limit
Host Computer : WinNT://WORKGROUP/VICKY
```

22.3 Enumerating Sessions and Resources

We now want to show you how to use ADSI to do the following:

- Enumerate a client's sessions and resources
- Show which users are currently logged on to a server and count all the logged-on users across a domain's PDCs, BDCs, and other servers

Windows NT, Windows 2000, and Windows Server 2003 machines host two kinds of dynamic objects that you can access with ADSI: sessions (i.e., instances of users connected to a computer) and resources (i.e., instances of file or queue access on a computer). When users connect to a file or a share on a computer, that creates both a session and a resource object. When the user disconnects, these dynamic objects cease to exist.

You can access dynamic objects by connecting directly to the Server service on the machine. Although each Server service has a user-friendly display name that appears in the Computer Management console in Windows 2000 and Windows Server 2003 or the Services applet in Control Panel in NT, each Server service also has an ordinary name that you use when connecting to it with ADSI. For example, Server is the display name of the service that has the short name LanManServer. If you enumerate all the services on a machine, you can use IADsService::DisplayName to print the display name and IADs::Name to print the short name.

LanManServer is an object of type FileService. FileService objects are responsible for maintaining the sessions and resources in their jurisdictions. You can use the IADsFileServiceOperations interface to access information about these sessions and resources. This simple interface has two methods: IADsFileServiceOperations::Sessions and IADsFileServiceOperations::Resources. Both methods return collections of objects that you can iterate through with a For Each...Next loop. When you're iterating through a collection in this manner, the system is using IADsCollection::GetObject to retrieve each item from the collection. As a result, you can use the same IADsCollection::GetObject method to retrieve a specific session or resource object. You then can use the IADsSession or IADsResource interface to manipulate that session or resource object's properties to access information. For example, if you retrieve a session object, you can access such information as the username of the user who is logged on and how long that user has been logged on.

22.3.1 Identifying a Machine's Sessions

The following script uses IADsSession to iterate through all the sessions on a particular machine:

```
On Error Resume Next

Dim objComputer, objSession, strOutput

strOutput = ""

Set objComputer = GetObject("WinNT://mydomainorworkgroup/mycomputer/LanManServer")

For Each objSession In objComputer.Sessions
  strOutput = strOutput & "Session Object Name : " & objSession.Name & vbCrLf
  strOutput = strOutput & "Client Computer Name: " & objSession.Computer & vbCrLf
  strOutput = strOutput & "Seconds connected   : " _
    & objSession.ConnectTime & vbCrLf
  strOutput = strOutput & "Seconds idle        : " & objSession.IdleTime & vbCrLf
  strOutput = strOutput & "Connected User      : " & objSession.User & vbCrLf
  strOutput = strOutput & vbCrLf
Next

WScript.Echo strOutput
```

This is straightforward. It uses the IADs::Name property method and IADsSession property methods to retrieve data about the session. The IADs::Name property method displays the object name, which is the name that you would use

22.4 Manipulating Print Queues and Print Jobs

So far we've shown you how to use ADSI to manipulate persistent and dynamic objects, such as shares, sessions, and resources. Now we're going to examine printer queues and jobs. In this section, we're going to lead you through creating scripts to do the following:

- Identify print queues in Active Directory
- Bind to a print queue[4] and access its properties

  [4] Print queues are logical ADSI names for printers installed on a computer.
- List the print jobs in a print queue and manipulate them

All the code in these scripts for managing printers is done using the WinNT provider, so it will work on Windows NT as well as Active Directory. The LDAP searches will not work on Windows NT.

One point before we go on: at the end of Chapter 20, we detail a function called SearchAD. We need to use it now to search Active Directory for the printer's ADsPath and store it in arrSearchResults(0,0).

22.4.1 Identifying Print Queues in Active Directory

List-Print-Queue.vbs in Example 22-2 is a heavily commented script, so it should be easy to follow.

Example 22-2. List-Print-Queue.vbs identifies print queues in Active Directory

```
 Option Explicit
On Error Resume Next

'*********************************************************************
'Active Directory path to start the search from
'*********************************************************************
Const strDomainToSearch = "LDAP://dc=mycorp,dc=com"

'*********************************************************************
'Maximizes the Notepad screen when started
'*********************************************************************
Const vbMaximizedFocus   = 3

'*********************************************************************
'Sets the location of the temporary file
'*********************************************************************
Const TEMPFILE = "C:\PRINTERLIST-TEMP.TXT"

'*********************************************************************
'Opens a file and lets you start writing from the beginning of the file
'*********************************************************************
Const ForWriting = 2

Dim arrPaths(  ), fso, ts, strItem, intRC, objShell, intIndex

If Not SearchAD(strDomainToSearch,"(objectClass=printQueue)","SubTree",arrPaths) Then
  MsgBox "Printer listing failed!"
Else
  '*********************************************************************
  'Opens the temporary text file for writing. If the text file already
```

22.5 Summary

While the future of automating systems management-related tasks lies with WMI, you can still use ADSI very effectively to accomplish a number of key tasks. In this chapter, we took a look at how you can use ADSI to manipulate persistent objects (like a computer's shares and services) and dynamic objects (computers' open resources, users' active sessions, and print jobs that users initiate) in Active Directory or Windows NT SAM.

Chapter 23. Permissions and Auditing

Security descriptors (SDs), access control lists (ACLs), and access control entries (ACEs) have been used for files and directories on NTFS filesystems for years. The same concepts apply to securing Active Directory objects as well. While the information in this chapter is focused on Active Directory, the principles of creating an SD that contains a discretionary access control list (DACL) and system access control list (SACL) can map exactly over to NTFS files and directories.

ADSI provides four main interfaces we can use:
IADsAccessControlEntry

Manipulates individual ACEs that represent access or audit permissions for specific users or groups to objects and properties in Active Directory.
IADsAccessControlList

Manages collections of ACEs for an object.
IADsSecurityDescriptor

Manages the different sets of ACLs to an object.
IADsSecurityUtility

Gets, sets, and retrieves security descriptors for an object.

All of the ADSI security interfaces can be found in the MSDN Library (http://msdn.microsoft.com/library/) under Networking and Directory Services ⟶ Active Directory, ADSI and Directory Services ⟶ SDK Documentation ⟶ Directory Services ⟶ Active Directory Service Interfaces ⟶ Active Directory Service Interfaces Reference ⟶ ADSI Interfaces ⟶ Security Interfaces.

Microsoft provides a DLL (ADsSecurity.dll) with the Platform SDK that contains several interfaces that you can use to manage security descriptors, ACLs, and ACEs. It isn't covered in this chapter because it doesn't come installed with Windows 2000 or Windows Server 2003, but we encourage you to check it out and take a look at the example source code that comes with it for more information. Remember that the DLL will need to be installed and registered using REGSVR32.EXE ADSecurity.dll on every client that would use it.

23.1 How to Create an ACE Using ADSI

Microsoft has a habit of calling a shovel a ground insertion earth management device, that is, they like to give names that are not always intuitive to the average person. The contents of the five properties of the ACE object are not all immediately obvious from the names. In addition, as Microsoft uses the ACE for system-audit and permissions entries, a number of values that can go into the properties make sense only in a particular context. To complicate matters further, one property (AceFlags) is a catchall area that currently is the location for two completely different sets of information.

Creating an ACE is a simple matter. To set up an ACE, you need the following basic pieces of information:
AccessMask

What permissions you want to set
AceType

Whether you are setting allow/deny permissions or auditing for an object or property
Trustee

Who to apply the permissions to
AceFlags

What inheritance options you want and, if it is an audit entry, whether you are monitoring successes or failures
Flags, ObjectType, InheritedObjectType

What the ACE applies to if not just the entire object

We will now go through several examples to show you what the five properties of an ACE will contain based on certain security settings. Let's start with the simple example: giving a user full control permissions to an Organizational Unit. That means the information in Table 23-1 gets stored as an ACE on the SD of the Organizational Unit itself.

Table 23-1. Contents of the ACE properties when giving a user full control permissions to an Organizational Unit

Name of the property	**Value to be stored**
Trustee	Names the user who is to have the permission.
AccessMask	Gives full control (i.e. give every permission).
AceType	This is an allow permission.
AceFlags	The permission applies to this object. Child objects inherit this ACE.
Flags	Neither ObjectType nor InheritedObjectType is set.
ObjectType	Null.
InheritedObjectType	Null.

The user (Trustee) is allowed (AceType) full control (AccessMask) to the current object and all objects down the tree (AceFlags). The last three are not used here, as the permission is a simple one to an entire object.

23.2 A Simple ADSI Example

All of the seven ACE properties are set using property methods of the same names as those in an ADSI interface called IADsAccessControlEntry. The ACEs that are created using this are then modified using IADsAccessControlList and IADsSecurityDescriptor.

Let's go through an example now so you can see how it all fits together. Example 23-1 shows a section of VBScript code that creates an ACE that allows ANewGroup full access to the myOU organizational unit and all its children.

Example 23-1. A simple ADSI example

```
'**************************************************************************
'Declare constants
'**************************************************************************
Const FULL_CONTROL = -1
Const ADS_ACETYPE_ACCESS_ALLOWED = 0
Const ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT = 2

'**************************************************************************
'Declare variables
'**************************************************************************
Dim objObject    'Any object
Dim objSecDesc   'SecurityDescriptor
Dim objDACL      'AccessControlList
Dim objNewACE    'AccessControlEntry

'**************************************************************************
'Create the new ACE and populate it
'**************************************************************************
Set objNewACE = CreateObject("AccessControlEntry")
objNewACE.Trustee = "AMER\ANewGroup"
objNewACE.AccessMask = FULL_CONTROL
objNewACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
objNewACE.AceFlags = ADS_FLAG_INHERITED_OBJECT_TYPE_PRESENT

'**************************************************************************
'Add the new ACE to the object and write it to the AD
'**************************************************************************
Set objObject = GetObject("LDAP://ou=myOU,dc=amer,dc=mycorp,dc=com")

'**************************************************************************
'Use IADs::Get to retrieve the SD for the object
'**************************************************************************
Set objSecDesc = objObject.Get("ntSecurityDescriptor")

'**************************************************************************
'Use IADsSecurityDescriptor:: DiscretionaryAcl to retrieve the existing DACL
'**************************************************************************
Set objDACL = objSecDesc.DiscretionaryAcl

'**************************************************************************
'Use IADsAccessControlList::AddACE to add an ACE to an existing DACL
'**************************************************************************
objDACL.AddAce objNewACE

'**************************************************************************
'Use IADsSecurityDescriptor:: DiscretionaryAcl to put back the modified DACL
'**************************************************************************
objSecDesc.DiscretionaryAcl = objDACL

'**************************************************************************
'Use IADs::Put to replace the SD for the object
'**************************************************************************
objObject.Put "ntSecurityDescriptor", Array(objSecDesc)

'..........................................................................
```

23.3 A Complex ACE Example

Example 23-2 shows two further ACEs being created. This time we have included all the constants. This example sets the following ACEs on myOU:

- No permissions even to see the object for members of DenyGroup.
- Ability to create, delete, and examine all children of the object for AllowChildGroup.
- Ability for user Vicky Launders to assume ownership of the Organizational Unit only and not any children.
- Permission for the user Lee Flight to read and write this OU's description.
- Permission for the Chris Heaton account to read and write all users' passwords
- Generation of audit messages for failed access by Everyone to delete the object itself.
- Generation of audit messages for all modifications to Active Directory by Brian Kerr below this Organizational Unit, but not including this Organizational Unit.

Example 23-2. A complex ACE example

```
 '************************************************************************
'AccessMask constants
'************************************************************************
Const ADS_RIGHT_GENERIC_READ = &H80000000
Const ADS_RIGHT_GENERIC_WRITE = &H40000000
Const ADS_RIGHT_GENERIC_EXECUTE = &H20000000
Const ADS_RIGHT_GENERIC_ALL = &H10000000
Const ADS_RIGHT_SYSTEM_SECURITY = &H1000000
Const ADS_RIGHT_SYNCHRONIZE = &H100000
Const ADS_RIGHT_WRITE_OWNER = &H80000
Const ADS_RIGHT_WRITE_DAC = &H40000
Const ADS_RIGHT_READ_CONTROL = &H20000
Const ADS_RIGHT_DELETE = &H10000
Const ADS_RIGHT_DS_CONTROL_ACCESS = &H100
Const ADS_RIGHT_DS_LIST_OBJECT = &H80
Const ADS_RIGHT_DS_DELETE_TREE = &H40
Const ADS_RIGHT_DS_WRITE_PROP = &H20
Const ADS_RIGHT_DS_READ_PROP = &H10
Const ADS_RIGHT_DS_SELF = &H8
Const ADS_RIGHT_ACTRL_DS_LIST = &H4
Const ADS_RIGHT_DS_DELETE_CHILD = &H2
Const ADS_RIGHT_DS_CREATE_CHILD = &H1
Const FULL_CONTROL = -1

'************************************************************************
'AceType constants
'************************************************************************
Const ADS_ACETYPE_SYSTEM_AUDIT_OBJECT = &H7
Const ADS_ACETYPE_ACCESS_DENIED_OBJECT = &H6
Const ADS_ACETYPE_ACCESS_ALLOWED_OBJECT = &H5
Const ADS_ACETYPE_SYSTEM_AUDIT = &H2
Const ADS_ACETYPE_ACCESS_DENIED = &H1
Const ADS_ACETYPE_ACCESS_ALLOWED = &H0
```

23.4 Creating Security Descriptors

If you are creating an object from scratch, and you don't want it to get the default DACL and SACL that due to inheritance would normally be applied to objects created at that location in the tree, you can write your own DACL and SACL for an object. As you would expect, there are a number of properties associated with security descriptors and ACLs that you need to set. SDs and ACLs can be manipulated with the IADsAccessControlList (see Table 23-10) and IADsSecurityDescriptor (see Table 23-11) interfaces. We'll go through these briefly now and then move on to some more examples.

Table 23-10. IADsAccessControlList methods and properties

IADsAccessControlList methods and properties	**Action**
AddAce method	Adds an ACE to an ACL
RemoveAce method	Removes an ACE from an ACL
CopyAccessList method	Copies the current ACL
AclRevision property	Shows the revision of the ACL (always set to 4; see later text)
AceCount property	Indicates the number of ACEs in the ACL

The revision level is a static version number for every ACE, ACL, and SD in Active Directory. It is defined in the ADS_SD_REVISION_ENUM enumerated type, which contains a single constant definition as follows:

```
Const ADS_SD_REVISION_DS = 4.
```

Having a revision allows Active Directory to know which elements of an ACE could exist. Later, if new properties and concepts are added to the ACE so that it has a more extended definition, the revision would increment. Active Directory would then know that old revision-4 ACEs could not support the new extensions and could upgrade them or support them with lesser functionality.

Table 23-11. IADsSecurityDescriptor methods and properties

IADsSecurityDescriptor methods and properties	**Action**
CopySecurityDescriptor method	A copy of an existing SD.
Revision property	The revision of the SD (always set to 4, as noted earlier).
Control property	A set of flags indicating various aspects of the SD (see later text).
Owner property	The SID of the owner. If this field is null, no owner is set.
OwnerDefaulted property	A Boolean value indicating whether the owner is derived by the default mechanism when created (i.e., assembled out of all the inherited ACEs passed down by its parents) rather than explicitly set by the person or application that

23.5 Listing ACEs to a File for All Objects in an OU and Below

A good example of a useful real-world task is when you are curious to see what ACEs have been set on all objects below a container, such as a domain or Organizational Unit. Example 23-4 is a piece of code that can be used as the basis for checking through an Active Directory forest looking for irregularities.

This code also could be used on the root of Active Directory when dealing with the problem outlined in Section 11.3.3 in Chapter 11. The code is fairly simple but very long, due to the fact that it has to check every constant for both the SACL and DACL of each object.

Example 23-4. Examining the ACEs on all objects below a container

```
On Error Resume Next

'**************************************************************************
'If the GUID corresponds to a schema object or attribute, then print the
'schema attribute/object name and the GUID. Otherwise just print the GUID.
'**************************************************************************
Sub PrintGUID(ByVal objType)

  Dim strACEGUID, bolFound, intIndex

  '**************************************************************************
  'Convert a GUID that starts and ends with {  } and has dashes within to a
  'simple string of text
  '**************************************************************************
  strACEGUID = Replace(Mid(objType,2,Len(objType)-2),"-","")

  '**************************************************************************
  'Scan the array of schema values for a matching GUID (after converting both
  'GUIDs to uppercase first). If a GUID is found, the name is printed.
  '**************************************************************************
  ts.WriteLine vbTab & vbTab & "GUID: " & objType
  For intIndex=0 To UBound(arrSchema,2)
    If (UCase(strACEGUID) = UCase(arrSchema(0,intIndex))) Then
      ts.WriteLine vbTab & vbTab & "Name: " & arrSchema(1,intIndex)
    End If
  Next
End Sub

'**************************************************************************
'This function checks to see if the first integer value contains the constant
'passed in as the second integer value. If it does, then the third parameter
'is written out to the file, and the first value is decremented by the amount
'of the constant.
'**************************************************************************
Sub CheckValue(ByRef lngValueToCheck, ByVal lngConstant, ByVal strConstantName)
  If ((lngValueToCheck And lngConstant) = lngConstant) Then
    ts.WriteLine vbTab & strConstantName
    lngValueToCheck = lngValueToCheck Xor lngConstant
  Else
    lngValueToCheck = lngValueToCheck
  End If
End Sub

'**************************************************************************
'AccessMask constants
'**************************************************************************
Const ADS_RIGHT_GENERIC_READ = &H80000000
Const ADS_RIGHT_GENERIC_WRITE = &H40000000
Const ADS_RIGHT_GENERIC_EXECUTE = &H20000000
Const ADS_RIGHT_GENERIC_ALL = &H10000000
Const ADS_RIGHT_SYSTEM_SECURITY = &H1000000
Const ADS_RIGHT_SYNCHRONIZE = &H100000
Const ADS_RIGHT_WRITE_OWNER = &H80000
```

23.6 Summary

This chapter took a very detailed look at the four main interfaces that you can use to manipulate and iterate over permissions and auditing entries for objects and attributes in your organization:

- IADsAccessControlEntry
- IADsAccessControlList
- IADsSecurityDescriptor
- IADsSecurityUtility

You should now have the tools in your programming belt necessary to modify the permissions in Active Directory as needed.

Chapter 24. Extending the Schema and the Active Directory Snap-Ins

This chapter takes a look at two different areas: programmatically extending the schema and customizing the functionality of the Active Directory administrative MMC snap-ins. While these topics may seem very different, they share the common thread of storing and presenting information beyond what Active Directory is configured to do by default. They are also related because you will often want to include new schema extensions in the Active Directory snap-ins.

In the first half of the chapter, we take a look at how you can manipulate the schema to include new attributes and classes. In the second half, we describe how to modify the various components of the Active Directory Users and Computers (ADUC) snap-in to include customized display names and menus. While we will focus on ADUC, the techniques presented in this chapter can be used to modify any of the Active Directory administrative snap-ins.

24.1 Modifying the Schema with ADSI

We've shown you how the schema works in Chapter 4, and how to design extensions in Chapter 12. Now let's take a look at how to query and manipulate the schema using ADSI.

24.1.1 IADsClass and IADsProperty

In addition to being able to query and update schema objects as you can any other type of object with the IADs interface, there are two main schema-specific interfaces available: IADsClass and IADsProperty. Each of these interfaces has a variety of useful methods and property methods to allow you to set mandatory properties for classes, optional properties for classes, maximum values for attributes, and so on. If you look at these interfaces, you will see that they are very simple to understand.

First, let's compare accessing and modifying the schema by using the attributes we are interested directly in versus using the IADsClass and IADsProperty methods. This first code section uses attributes directly:

```
 objAttribute.Put "isSingleValued", False
objAttribute.Put "attributeId", "1.3.6.1.4.1.999999.1.1.28"

arrMustContain = objSchemaClass.Get("mustContain")
arrMayContain = objSchemaClass.Get("mayContain")
```

Now we will use the ADSI schema interfaces to do the same thing:

```
 objAttribute.MultiValued = True
objAttribute.OID = "1.3.6.1.4.1.999999.1.1.28"

arrMustContain = objSchemaClass.MandatoryProperties
arrMayContain = objSchemaClass.OptionalProperties
```

This makes use of IADsProperty::MultiValued, IADsProperty::OID, IADsClass::MandatoryProperties, and IADsClass::OptionalProperties. As you can see, it's not hard to convert the code. However, we feel that including code that directly modifies the properties themselves gives you some idea of what you are actually changing and helps you to refer back to the definitions presented in Chapter 4.

More details on these three interfaces can be found in the MSDN Library (http://msdn.microsoft.com/library/) under Networking and Directory Services ⟶ Active Directory, ADSI and Directory Services ⟶ SDK Documentation ⟶ Directory Services ⟶ Active Directory Service Interfaces ⟶ ADSI Reference ⟶ ADSI Interfaces ⟶ Schema Interfaces.

24.1.2 Creating the Mycorp-LanguagesSpoken attribute

We will create an example attribute called Mycorp-LanguagesSpoken. It is to be a multivalued, indexed attribute that can hold an array of case-sensitive strings of between 1 and 50 characters. The name is prefixed with Mycorp so it is obvious that Mycorp created the attribute.

Mycorp's Schema Manager has decided that the OID for this attribute is to be 1.3.6.1.4.1.999999.1.1.28. This is worked out as follows:

- Mycorp's root OID namespace is 1.3.6.1.4.1.999999.
- Mycorp's new attributes use 1.3.6.1.4.1.999999.1.1.xxxx (where xxxx increments from 1).
- Mycorp's new classes use 1.3.6.1.4.1.999999.1.2.xxxx (where xxxx increments from 1).
-

24.2 Customizing the Active Directory Administrative Snap-ins

For those who have worked with Windows NT domains, you are undoubtedly familiar with two GUI tools: User Manager (*usrmgr.exe*) and Server Manager (*srvmgr.exe*). User Manager allows administrators to manipulate the properties of users and groups, while Server Manager can manipulate computer accounts. In Active Directory, a Microsoft Management Console (MMC) snap-in called Active Directory Users and Computers (ADUC) has taken the place of both these tools.

While ADUC is built primarily to manage users, groups, and computers as the previous User Manager and Server Manager did, you can actually use it to manage any type of object within a Domain Naming Context. You can create an entire hierarchy of Organizational Units, user accounts, computer accounts, groups, printers, and so on and manage them with ADUC. The tool, however, is limited in what it provides "out of the box." While ADUC can display a lot of attributes for objects, you cannot view every attribute, as you can with ADSI Edit. Figure 24-3 shows the various groupings of attributes (e.g., Organization) that can be viewed by clicking the appropriate tab. Each tab represents a property page, which contains a logical grouping of attributes to display.

Figure 24-3. Numerous property pages for a user object

Now compare Figure 24-3 with Figure 24-4, which shows the property pages for a computer object.

Figure 24-4. Significantly fewer property pages for a computer obj

24.3 Summary

In this chapter, we covered how to query and manipulate the Active Directory Schema, including how to locate and transfer the Schema FSMO. The schema cache and its importance was also briefly touched on, along with information on how to determine which attributes of an object are in the GC and how to add an attribute to the GC if necessary.

The second part of the chapter focused on how to customize the Active Directory administrative MMC snap-ins by modifying displaySpecifier objects. We described how to manipulate each of the major snap-in components, including property pages, context menus, icons, display names and the object creation wizard. For more information about customizing snap-ins, check out the following locations in the MSDN Library (http://msdn.microsoft.com/library/):

- Networking and Directory Services ⟶ Active Directory, ADSI, and Directory Services ⟶ SDK Documentation ⟶ Directory Services ⟶ Active Directory ⟶ Using Active Directory ⟶ Extending the User Interface for Directory Objects
- Networking and Directory Services ⟶ Active Directory, ADSI, and Directory Services ⟶ SDK Documentation ⟶ Directory Services ⟶ Active Directory ⟶ Active Directory Reference ⟶ Active Directory Interfaces ⟶ Active Directory Admin Interfaces
- User Interface Design and Development ⟶ Windows Shell ⟶ Shell Programmers Guide

Chapter 25. Using ADSI and ADO from ASP or VB

Two important features of Active Directory require administrators to create their own tools:

- The ability to extend the Active Directory schema with your own classes and attributes, which allows you to store additional data with objects
- The ability to delegate control of administration of Active Directory in a very detailed manner

If you take advantage of these, there is a large chance that you will want to provide customized tools for administration.

For example, you might decide that a group of users is to manage only certain properties of certain objects, say which users can go into a group. There is no point in giving them Active Directory Users and Computers snap-in; that's like using a sledgehammer to crack a nut. Why not create a tool of your own that only allows them to manipulate the values that they have permission to? If you then incorporate logging into a file or database within this application, you have a customized audit trail as well.

Tools of this nature do not lend themselves to VBScript since they tend to require a much more enhanced GUI interface. Consequently, you are left with three choices:

- Write code in a compiled language like Visual Basic or VB.NET that supports complex GUI routines.
- Write code for a web-based interface using HTML and Active Server Pages (ASPs) or using ASP.NET.
- Write code in another scripting language such as Perl that supports complex graphical controls.

We will concentrate on the first two in this chapter.

25.1 VBScript Limitations and Solutions

Using ADSI from within WSH is very useful, but it does have certain limitations. For one thing, you cannot display output on screen in anything other than a MsgBox or request information from users without using the InputBox. It is easy to show how these are lacking. Consider that we wish to write a general script that adds a user to a single group selected from a list. If we wrote this under WSH, we would have to list all the groups to the screen in a large MsgBox (or via a file using Notepad) with incremental numbers so that each group could be identified. Then the person running the script would have to remember the number and type it into an InputBox later so that the request could be serviced. If there were more than a few dozen groups in Active Directory, the person running the script would have to go through a number of screens of groups before being able to see them all. It would be much simpler just to display a drop-down list box of all groups and have the user select one. This is not possible under WSH using VBScript, but it is possible under VB and Active Server Pages (ASP).

VB provides a full programming environment for your ADSI applications. ASP provides VBScript with the user-interface facilities that HTML allows, effectively making your scripts more user friendly. ASPs are useful for two important reasons. First, there is a single copy maintained in the organization. Hence, if the single copy is updated, everyone gets the latest copy on the next use. This also saves you from version hell—having multiple versions of a program floating around. Second, no runtime or design-time licenses are required in the development of such pages, as is the case when you develop VB applications.

Also, if we publish the web pages on an Internet server rather than an intranet server, we can make the scripts available to anyone who has the correct privilege to the script whether he is on our local network or not. At present you may find it hard to see a need for being able to manipulate Active Directory from outside the organization. As Active Directory becomes a larger store for complex objects, you may find yourself writing pages to interrogate company databases as well as Active Directory, bringing both sets of information forward to the user. Web pages also allow you to prototype or identify a need for a future application. If you find that your users are making heavy use of the web interface, perhaps it is time to consider rolling out a proper application. It all depends on what sort of mechanism you prefer to develop and maintain to let your users access your Active Directory.

This chapter will describe in detail how to create ASPs using HTML and ADSI and how to migrate VBScript scripts to simple VB applications.

While incorporating ADSI scripts into ASPs via HTML is fairly easy, anyone who is considering using VBScript with HTML pages needs to do some background reading. This chapter alone barely scratches the surface and in no way covers HTML in any real depth.

25.2 How to Avoid Problems When Using ADSI and ASP

There is one very large pitfall with ADSI scripts under ASP that is very easy to fall into. ADSI scripts running under ASP work only when served from IIS. This is because IIS understands ADSI, and IE on its own does not. So whenever you want to test-run an ASP incorporating an ADSI script, make sure that you are obtaining it from the server. This problem tends to occur in two main ways:

- When developing scripts on the machine that IIS is running on
- When developing scripts on a machine that has a drive mapped to the directory on IIS where you are storing the scripts

In both of these cases, it is just as easy to open a file called *C:\INETPUB\WWWROOT\MYTEST.ASP* as it is to open http://www.mycorp.com/mytest.asp from within IE. Both files will open correctly, but only the IIS-served page will correctly work with ADSI. If you start getting unexplained errors with code that you know should be working, just check the URL of the ASP that you are opening.

The second annoying pitfall occurs when you are constantly updating pages, testing them with a browser, and then updating them again. If you are developing in this cycle, remember to keep refreshing the page. It becomes really annoying to find that the bug you have been trying to solve is due to the fact that your browser thoughtfully cached a page 15 minutes ago, and you have been forgetting to press the Shift key when clicking the Refresh button.[1]

[1] Another option if you are using Internet Explorer is to open up the Internet Options from the Tools menu and set the Temporary Internet Files to check for newer versions of stored pages on every visit to the page.

25.3 Combining VBScript and HTML

HTML pages are written as text files in the same way as VBScripts. HTML pages display information according to a series of tags, which you use to turn certain formatting on and off. The tags you normally use to construct a basic page look like this:

```
 <HTML>

<HEAD>
<TITLE>Hello World Page</TITLE>
</HEAD>

<BODY>
Hello World
<P>Hello again
</BODY>

</HTML>
```

The <HTML> tag denotes it as an HTML document. The <HEAD> and </HEAD> pair denote everything within those tags as belonging to the header description part of the document. The <TITLE> tag denotes the start of the page title and </TITLE> turns it off again. The <BODY> tag denotes the main part of the page, which contains a simple text string, a newline or paragraph marker <P>, and then another line of text. This is the bare bones of writing HTML documents. You can create lists, set colors, make and populate tables, display images, and so on. However, you do not need to go into all of that to demonstrate incorporating ADSI VBScripts into HTML pages. You only need to be aware of the following major sets of tags: <FORM> . . . </FORM>, <OBJECT> . . . </OBJECT>, <% . . . %>, and <SCRIPT> . . . </SCRIPT>.

25.3.1 Incorporating Scripts into Active Server Pages

Two sorts of scripts can be created within ASPs: client-side scripts and server-side scripts. Client-side scripting is used to access all the objects in a web page (e.g., text, images, tags), browser objects (e.g., frames, windows), and local ActiveX components. Server-side scripting is used to create a web page dynamically via parameters, forms, and code that is then passed to a browser.

Because the two types of scripts are executed at different locations, each has a separate set of interfaces. You place your ADSI scripts in server-side scripting, not client-side scripting. We'll go through the major differences now so that you will be less likely to make annoying mistakes.

25.3.1.1 Client-side scripting

You can use the <SCRIPT> tags to add client-side VBScript code to an HTML page. Whenever the browser encounters the tags, the enclosed script is executed as if it were being issued from the client. You can use blocks of scripting in both the BODY and HEAD sections of an ASP if you want to. If you put your code in the HEAD section, it will be read and executed before any item in the BODY section is accessed. As an example, here is a procedure to display a line of text:

```
 <SCRIPT LANGUAGE="VBScript">

  Document.Write "This is a line of text<P>"

</SCRIPT>
```

The LANGUAGE attribute indicates that this is VBScript rather than one of the other languages. As this is not running under the WSH, you do not have a VBS or JS extension to denote the language. The Document::Write method writes the line to the web page. It is only one of a number of properties and methods from interfaces available to you as an ASP developer. You also can use MsgBox and InputBox within client-side scripts.

The important thing about client-side scripts from this chapter's point of view is that ADSI functions and methods cannot be included in these scripts. This is an important limitation, one that we will show you how to get around later.

25.4 Binding to Objects Via Authentication

Whenever we need to access the properties of an object in Active Directory, we bind to it using VBScript's GetObject function or the ADSI method IADsOpenDSObject::OpenDSObject. The circumstances in which each method should be used to access Active Directory is very clear-cut but deserves to be outlined here, as it will be important whenever you construct ASPs.

25.4.1 When to Use VBScript's GetObject Function

By default, many of the objects and properties within Active Directory can be read by any authenticated user of the forest. As an example, here is some code to connect to an Organizational Unit called Sales under the root of the domain. This code works under the WSH:

```
 Set objSalesOU = GetObject("LDAP://ou=Sales,dc=mycorp,dc=com")
Wscript.Echo objSalesOU.Description
```

Here is the same script incorporated into an ASP:

```
 <HTML>
<HEAD>
<TITLE>Binding to an existing Organizational Unit</TITLE>
</HEAD>

<BODY>
<%
  Set objSalesOU = GetObject("LDAP://ou=Sales,dc=mycorp,dc=com")
  Response.Write "The Sales OU description is: " & objSalesOU.Description
%>
</BODY>
</HTML>
```

This mechanism works perfectly when you wish to have read-only access to properties of objects that can be read without special privileges. Using GetObject is not appropriate in the following cases:

- You want to write properties of an object.
- The object you are attempting to bind to requires elevated privileges to access.

While it may make little sense, it is perfectly feasible to restrict read access to the description of the Sales Organizational Unit, or more commonly the Sales Organizational Unit itself. If the Sales Organizational Unit is restricted, a GetObject will fail to bind to it. If only the description is restricted, a GetObject will successfully bind to the Sales Organizational Unit, but access to the description property will be denied.

To gain access to a restricted object or impersonate another user, you must authenticate using IADsOpenDSObject::OpenDSObject.

25.4.2 When to Use IADsOpenDSObject::OpenDSObject

Here is a simple Organizational Unit creation script that works under the WSH when an administrator is logged in:

```
 Set objRoot=GetObject("LDAP://dc=mycorp,dc=com")

Set objSalesOU = objRoot.Create("organizationalUnit","ou=Sales")
objSalesOU.Description = "My new description!"
objSalesOU.SetInfo
```

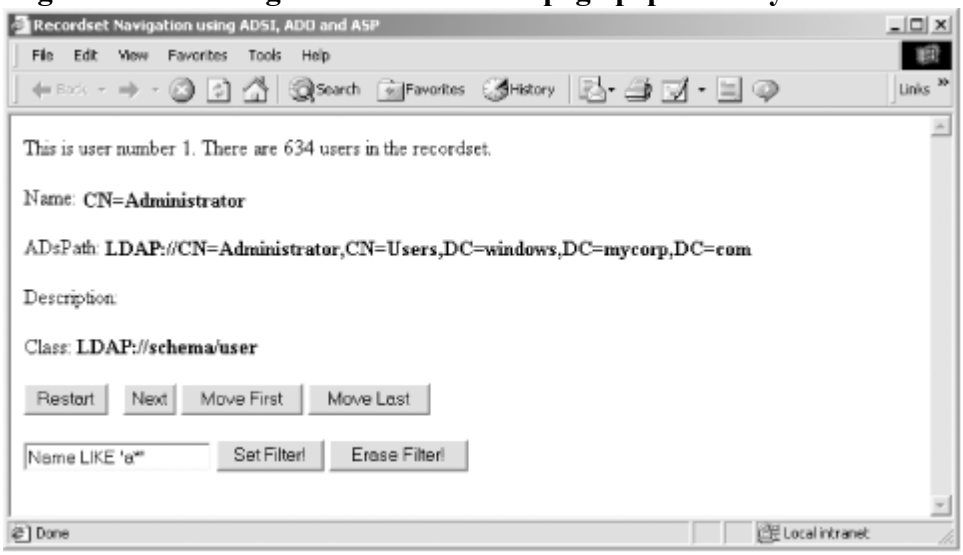
We cannot transfer the script to an ASP as it stands. To make the script work, we must use the IADsOpenDSObject::OpenDSObject method, which does allow authentication. Here is the same example using authentication within an ASP:

```
 <HTML>
```

25.5 Incorporating Searches into ASP

ADO searches can be easily incorporated into ASPs using the information in this chapter and Chapter 20. In this first example, we will navigate through a resultset using server-side scripts in order to populate a table that gets created dynamically. To make it easier to understand, Figure 25-1 is what the final result should look like for a new server with very few users.

Figure 25-1. A navigable table on a web page populated by ADO



This ASP includes all its code in the body of the web page. To begin with, we must retrieve the resultset:

```
<%
  Set objConn = CreateObject("ADODB.Connection")
  objConn.Provider = "ADSDSOObject"
  objConn.Open "", _
    "CN=Administrator,CN=Users,dc=mycorp,dc=com", ""

  Set objRS = objConn.Execute _
    ("<LDAP://dc=mycorp,dc=com>;" _
      & "(objectClass=User);Name,ADsPath;SubTree")
%>
```

Having done this, we can now begin to create the table. The table definition must include the number of columns. Even though we know that we are retrieving two columns, we will include the value returned from the query rather than hardcoding a value of 2 into the table so that we can extend the page later. The table definition then looks like this:

```
<TABLE BORDER=1 COLS=<% = objRS.Fields.Count%>>
```

Now we need to include column headings. Again, if we take these directly from the query, then we can expand the query much more easily later:

```
   <TR>
    <% For Each adoField In objRS.Fields %>
         <TH> <% = adoField.Name %> </TH>
    <% Next %>
  </TR>
```

Now we can navigate through the actual resultset and populate the table. Each row is created via the <TR>...</TR> pair of tags by navigating through the resultset using a Do While...Loop construct. As soon as we go past the end of the loop, the table closing tag is sent. Each individual row is populated using a For...Each loop:

```
   <% Do While Not objRS.EOF %>
    <TR>
      <% For Each adoField In objRS.Fields %>
         ' Populate the cells here
      <% Next
      objRS.MoveNext %>
    </TR>
```

25.6 Migrating Your ADSI Scriptsfrom VBScript to VB

If you decide you need a GUI-based application instead of a web-based application, it is time to start thinking about coding in a different language. VB is an easy language with the capabilities of great complexity. The VB language itself is very similar to VBScript, so you can port code quickly from your existing scripts. However, there is so much that you can do with VB that the bewildering array of interfaces and methods can easily get confusing. The simplest solution to this is to get a book on VB. There are many dozens of books that already exist on the complexities of writing in VB, and we do not intend to provide an introduction here. If you are seriously considering writing in VB, your best option is to pick up a book on it.

This section covers what you need to do to write ADSI code with VB after having written ADSI code in VBScript. This includes a brief look at the major differences between VBScript and VB, the options that need to be set, and the Platform SDK, which you will need to compile your code. We also briefly cover a series of examples that are available from the O'Reilly web site. The notes that we present in this section are with respect to Microsoft Visual Basic Professional Version 6.0. However, these examples should also work with future versions of VB as well.

25.6.1 Platform Software Development Kit

To access the ADSI interfaces and libraries, you need to be able to reference the appropriate component of the Microsoft Platform Software Development Kit (SDK) in your code. You can either download the appropriate component or obtain the full SDK, which includes all components.

The full SDK provides developers with a single, easy-to-use location from which to download current and emerging Microsoft technologies; it includes tools, headers, libraries, and sample code. The Platform SDK is the successor to the individual SDKs, such as the Win32, BackOffice, ActiveX/Internet Client, WMI, ADSI, and DirectX SDKs.

You can get the full SDK build environment or just the ADSI component in a number of ways:

- If you purchase an MSDN Professional-level subscription, you will be shipped all of the SDKs that you require.
- If you purchase an MSDN Enterprise-level subscription, you will be shipped all of the SDKs and all of the Visual Studio products, which includes Microsoft Visual Basic Enterprise Edition as well.
- If you purchase Visual Basic 6.0 Enterprise Edition, you receive the full MSDN set of CDs and the SDK build environment.

You can download the parts of the platform SDK by following going to http://www.microsoft.com/msdownload/platformsdk/sdkupdate/.

If you wish to make use of ADO from the next chapter, you need Microsoft Data Access Components (MDAC) as well. You can download these from the Downloads section of the Universal Data Access site: http://www.microsoft.com/data/download.htm.

Once the SDK has been downloaded and installed, start VB and in any new project that you write make sure that you go to Project ⟶ References and check items according to Table 25-1.

Table 25-1. When to use relevant references in VB

Reference	To use

25.7 Summary

Being able to customize the Active Directory schema means that you may end up using a number of new classes and attributes that you create. As these classes and attributes can be manipulated using the same ADSI interfaces that you have seen in the previous chapters, you can easily create your own customized tools to operate on these new objects. This allows you free rein in developing solutions that are perfectly tailored for your requirements, whether from a web-based or GUI-based interface.

Chapter 26. Scripting with WMI

The Windows Management Instrumentation (WMI) API was developed by Microsoft in 1998 in response to the ever-growing need for developers and system administrators to have a common, scriptable API to manage the components of the Windows operating systems. Before WMI, if you wanted to manage some component of the operating system, you had to resort to using one of the component specific Win32 API's, such as the Registry API or Event Log API. Each API typically had its own implementation quirks and required way too much work to do simple tasks. The other big problem with the Win32 APIs is that scripting languages such as VBScript could not use them. This really limited how much an inexperienced programmer or system administrator could do to programmatically manage systems. WMI changes all this by providing a single API that can be used to query and manage the Event Log, the Registry, processes, the filesystem, or any other operating system component.

So you may be wondering at this point: this is a book on Active Directory, so why do I need to care about a system management API? Even if your sole job in life is to manage Active Directory, WMI can benefit you in at least two ways. First, Active Directory runs on top of Windows 2000 or Windows Server 2003. These servers need to be managed (i.e., Event Log settings configured, Registry modified, applications installed, etc.) and monitored (i.e., filesystem space, services running, etc.). You can choose to do all of those tasks manually, or you can use WMI to automate them. For each task you automate, the total cost of ownership to support Active Directory is reduced, and you help ensure your servers stay consistent. The other reason why WMI is important to Active Directory is the direction Microsoft is taking WMI with respect to monitoring and managing any system or application under the Microsoft umbrella. That's right, not only does Microsoft want WMI to be the primary interface to manage and monitor Windows systems, but also any Windows application, including Active Directory. Currently, ADSI provides the primary management interface into Active Directory, but in the Windows Server 2003 release, there are several new WMI hooks into Active Directory to monitor things such as trusts and replication.

In this chapter, we will give a brief introduction to the concepts and terminology behind WMI and then delve into several sample scripts showing how to make use of it. We will cover some system-specific tasks, such as managing services, the Event Log, and the Registry, which should give you a good grounding in some of the fundamentals of WMI. In the second half of the chapter, we will review how WMI can be used to access and monitor Active Directory.

In a single chapter we can only go into so much detail about the internals of WMI. We won't be covering some of the more advanced topics. If you are interested in more information than what this chapter provides, we recommend checking out the MSDN Library or one of the WMI books available on the market. At the time this book was published, you could access the WMI SDK documentation by going to the MSDN Library (http://msdn.microsoft.com/library) and visiting Setup and System Administration ⟶ Windows Management Instrumentation (WMI) ⟶ SDK Documentation or by going to the following web page: http://msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_start_page.asp.

26.1 Origins of WMI

There have been several industry initiatives over the years to develop a model for managing systems and devices that would be robust enough to meet the needs of most vendors. Several protocols and frameworks have been developed to address the problem. The Simple Network Management Protocol (SNMP) is probably the most notable, but is pretty simple in its implementation and does not provide many features most vendors need for a single management framework.

The Distributed Management Task Force (DTMF) was created in the early 1990s to address the management framework problem. They developed the Web Based Enterprise Management (WBEM) standard, which attempts to unify the management frameworks utilizing web technologies. As part of the WBEM standard, they also created the Common Information Model (CIM), which is the language used for describing management data in an object-oriented way. The WBEM/CIM standards have garnered a lot of industry support in recent years and provide the basis for WMI.

For more information on WBEM/CIM, check out the DMTF website: http://www.dmtf.org.

26.2 WMI Architecture

The WMI architecture is composed of two primary layers: the CIM infrastructure, which includes the CIMOM and CIM Repository, and the WMI providers. While the concepts Microsoft uses are very similar to the WBEM/CIM standards, they did not implement one very important component: the use of web technologies for the transport mechanism. Instead of using HTTP to transport messages between the WMI infrastructure and clients, Microsoft uses COM and DCOM, two Microsoft-specific technologies. This limits the use of WMI to only Microsoft platforms.

That being said, the capabilities to manage Microsoft-based platforms with WMI are nearly unlimited. More and more vendors are utilizing WMI not only to manage components of the Microsoft OS but also to manage their own applications. NMicrosoft has also become heavily invested in WMI by providing WMI providers for nearly all of its major applications, including Active Directory, Exchange 2000, DNS, and even Microsoft Office.

26.2.1 CIMOM and CIM Repository

The CIM Repository is the primary warehouse for management data. It contains the static data that does not change very frequently, such as memory or disk size. The CIMOM or CIM Object Manager handles requests from clients, retrieves data from the CIM Repository, and returns it to the client. The CIMOM also provides an event service, so that clients can register for events and be notified dynamically when they occur. For dynamic data, such as performance monitor counters, the CIMOM will interact directly with a WMI provider instead of retrieving the data directly from the CIM Repository. The CIM Repository cannot store all possible data that is needed by the various WMI providers. The storage requirements would be significant, not to mention that a lot of the data would become out-of-date almost immediately after it was stored.

26.2.2 WMI Providers

The WMI providers contain much of the intelligence behind WMI. Typically a provider will be implemented for each individual managed component, such as the Event Log or Active Directory Trusts. Each provider is responsible for interacting with its managed component and can perform certain functions implemented by methods on classes representing that component. Also, as described earlier, some providers interact with the CIMOM to provide dynamic data that cannot be held in the CIM Repository.

Each WMI provider is also associated with a namespace. The namespace is used to segregate where WMI providers store their data and class definitions. Think of it as a file system. You could store all of your files in a single directory, but it would be hard to manage. By storing data and class definitions for providers under different namespaces, you don't have to worry about confusing the EventLog provider with the Active Directory Trust provider. Table 26-1 contains the more commonly used and AD-related WMI providers and the associated namespace.

Table 26-1. Some of the commonly used and AD-related WMI providers

Provider	**Namespace**
Win32 provider	root\cimv2
EventLog provider	root\cimv2
Registry provider	root\default
Active Directory provider	root\directory\LDAP
Replication provider	root\MicrosoftActiveDirectory

26.3 Getting Started with WMI Scripting

Once you have a basic understanding of the WMI architecture, scripting with WMI is easy. In fact, once you understand how to reference, enumerate and query objects of a particular class with WMI, it is straightforward to adapt the code to work with any managed component.

26.3.1 Referencing an Object

To reference objects in WMI, you use a UNC-style path name. An example of how to reference the C: drive on a computer looks like the following:

```
\\dc1\root\CIMv2:Win32_LogicalDisk.DeviceID="C:"
```

The format should be easy to follow. The first part of the path (\\dc1\) is a reference to the computer on which the object resides. To reference the computer on which the script is running, you can use a "." for the computer name. The second part (root\CIMv2) is the namespace the object resides in. The third part (Win32_LogicalDisk) is the class of the object to reference. The fourth part is the key/value pairs representing the object. Generically, the path can be shown as follows:

```
\\ComputerName\NameSpace:ClassName.KeyName="KeyValue"[,KeyName2="KeyValue2"...]
```

Now that we know how to reference WMI objects, let's go ahead and instantiate an object using VBScript's GetObject function. For GetObject to understand that we are referencing WMI objects, we have to include one additional piece of information: the moniker. Just as we've been using the LDAP: and WinNT: progIDs to reference Active Directory and SAM-based objects in ADSI, we need to use the winmgmts: moniker when we are dealing with WMI objects:

```
Set objDisk = GetObject("winmgmts:\\dc1\root\CIMv2:Win32_LogicalDisk.DeviceID='C:'")
```

Note that if you want to reference the *C*: logical drive on the local computer, you can leave off the computer name and namespace path. The GetObject call would then look like this:

```
Set objDisk = GetObject("winmgmts:Win32_LogicalDisk.DeviceID='C:'")
```

You can leave out the namespace path because root\CIMv2 is the default namespace. When accessing a provider that uses any other namespace, you need to include the namespace path. Also, if you are referencing a remote object, you need to include the namespace path even if it is root\CIMv2.

26.3.2 Enumerating Objects of a Particular Class

Now let's look at an example. We want to view all logical disks on a machine, not just a particular disk. To do so, we need to use the InstancesOf method on a WMI object pointing to the namespace of the provider that contains the class. Perhaps an example will make this clear:

```
 strComputer = "."
Set objWMI = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set objDisks = objWMI.InstancesOf("Win32_LogicalDisk")
for each objDisk in objDisks
    Wscript.Echo "DeviceID: " &  objDisk.DeviceID
    Wscript.Echo "FileSystem: " &  objDisk.FileSystem
    Wscript.Echo "FreeSpace: " & objDisk.FreeSpace
    Wscript.Echo "Name: " & objDisk.Name
    Wscript.Echo "Size: " & objDisk.Size
    WScript.Echo ""
next
```

Here we get a WMI object pointing to the root\CIMv2 namespace, after which we call the InstancesOf method and pass the Win32_LogicalDisk class. That method returns a collection of Win32_LogicalDisk objects which we then iterate over with a For Each loop.

26.4 WMI Tools

There are several tools available to query and browse WMI information. These tools can be very useful in situations in which you want to access WMI information but do not want to write a script to do it.

26.4.1 WMI from a Command line

The WMI command-line tool (WMIC) is a powerful tool that can expose virtually any WMI information you want to access. It is available in Windows XP and Windows Server 2003. Unfortunately, WMIC does not run on Windows 2000, but it can still be used to query WMI on a Windows 2000 machine.

WMIC maps certain WMI classes to "aliases." Aliases are used as shorthand so that you only need to type "logicaldisk" instead of "Win32_LogicalDisk". An easy way to get started with WMIC is to type the alias name of the class you are interested in. A list of all the objects that match that alias/class will be listed.

```
 wmic:root\cli>logicaldisk list brief
DeviceID  DriveType  FreeSpace    ProviderName  Size         VolumeName
A:        2
C:        3          1540900864                 4296498688   W2K
D:        3          15499956224                15568003072
Z:        5          0                          576038912    NR1EFRE_EN
```

Most aliases have a list brief subcommand that will display a subset of the properties for each object. You can run similar queries for services, CPUs, processes, and so on. For a complete list of the aliases, type alias at the WMIC prompt.

The creators of WMIC didn't stop with simple lists. You can also utilize WQL to do more complex queries. This next example displays all logical disks with a drivetype of 3 (local hard drive):

```
 wmic:root\cli>logicaldisk where (drivetype = '3') list brief
DeviceID  DriveType  FreeSpace    ProviderName  Size         VolumeName
C:        3          1540806144                 4296498688   W2K
D:        3          15499956224                15568003072
```

We have just touched the surface of the capabilities of WMIC. You can invoke actions, such as creating or killing a process or service, and modify WMI data through WMIC as well. For more information, check out the Support WebCast "WMIC: A New Approach to Managing Windows Infrastructure from a Command Line," available at http://support.microsoft.com/default.aspx?scid=/webcasts/. Help information is also available on Windows XP and Windows Server 2003 computers by going to Start ⟶ Help, and search on WMIC.

26.4.2 WMI from the Web

Included as sample applications with the original WMI SDK, the WMI CIM Studio and WMI Object browser are web-based applications that provide much more benefit than just being example applications provided in the SDK. The following is a list of the tools and their purpose:

- The WMI CIM Studio is a generic WMI management tool that allows you to browse namespaces, instantiate objects, view the instances of a class, run methods, edit properties, and even perform WQL queries.
- The WMI Object Browser allows you to view the properties for a specific object, look at the class hierarchy, view any associations, run methods, and edit properties for an object.
- The WMI Event Registration allows you to create, view, and configure event consumers.
-

26.5 Manipulating Services

Querying services is simple to do with WMI. The Win32_Service class is the WMI representation of a service. The Win32_Service class contains a lot of property methods that provide information about the service; the most useful ones have been listed in Table 26-2.

Table 26-2. Useful Win32_Service properties

Property	Description
AcceptPause	Returns a Boolean indicating whether the service can be paused.
AcceptStop	Returns a Boolean indicating whether the service can be stopped.
Description	Description of the service.
DisplayName	Display name of the service.
Name	Unique string identifier for the service.
PathName	Fully qualified path to the service executable.
Started	Boolean indicating whether the service has been started.
StartMode	String specifying the start mode of the service. Will be one of Automatic, Manual, or Disabled.
StartName	Account under which the service runs.
State	Current state of the service. Will be one of Stopped, Start Pending, Stop Pending, Running, Continue Pending, Pause Pending, Paused, or Unknown.

The following script retrieves all the running services on a machine. All we need to do is use a WQL query that finds all Win32_Service objects that have a state of "Running":

```
 strComputer = "."

Set objWMI = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set objServices = objWMI.ExecQuery _
          ("SELECT * FROM Win32_Service WHERE State = 'Running'")
For Each objService in objServices
   Wscript.Echo objService.DisplayName
   Wscript.Echo " Name: " & objService.Name
   Wscript.Echo " PathName: " & objService.PathName
   Wscript.Echo " Started: " & objService.Started
   Wscript.Echo " StartMode: " & objService.StartMode
   Wscript.Echo " StartName: " & objService.StartName
   Wscript.Echo " State: " & objService.State
   Wscript.Echo ""
next
```

26.6 Querying the Event Logs

The Event Logs are typically a system administrator's first line of inquiry when trying to troubleshoot problems. Since they are so important, it is also important to see how we can make use of them with WMI. The two major components that we need to be concerned with are the Event Logs themselves and the events contained within each Event Log. We will first focus on properties of Event Logs.

The Win32_NTEventLogFile class represents an Event Log. Table 26-4 contains several Win32_NTEventLogFile properties that can be used to query or modify properties of a Event Log.

Table 26-4. Useful Win32_NTEventLogFile properties

Property	Description
FileSize	Size of the Event Log file in bytes.
LogFileName	Standard name used for describing the Event Log (e.g., Application).
MaxFileSize	Max size in bytes that the Event Log file can reach. This is a writeable property.
Name	Fully qualified path to the Event Log file.
NumberOfRecords	Total number of records in the Event Log.
OverwriteOutDated	Number of days after which events can be overwritten. This is a writeable property with 0 indicating to overwrite events as needed, 1-365 being the number of days to wait before overwriting, and 4294967295 indicating that events should never be overwritten.
OverwritePolicy	Text description of the overwrite policy (as specified by the OverwriteOutDated property). Can be one of WhenNeeded, OutDated, or Never.
Sources	Array of registered sources that may write entries to the Event Log.

Let's look at an example that displays all of the properties listed in Table 26-4 for each Event Log and sets the MaxFileSize and OverwriteOutDated properties if they have not already been set to the correct values. Since we want to iterate over all Event Logs, we will pass Win32_NTEventLogFile to the InstancesOf method. Example 26-2 shows how to accomplish this.

Example 26-2. Displaying properties of the Event Log using Win32_NTEventLogFile

```
 strComputer = "."
intMaxFileSize = 10 * 1024 * 1024    ' << 10MB
intOverwriteOutDated = 180           ' << 6 months

Set objWMI = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
Set objELF = objWMI.InstancesOf("Win32_NTEventLogFile")
' Iterate over each Event Log
```

26.7 Querying AD with WMI

Up to now, we've shown how WMI can be a powerful resource to aid in managing components of individual computers. You may be wondering what impact WMI will have on Active Directory? It can, in fact, play as big a role in automating the management of Active Directory as you want. Also, over time, WMI's importance with respect to monitoring Active Directory will continue to grow as Microsoft develops new providers.

First we are going to review how you can use WMI and the Active Directory provider to access and query objects in Active Directory. We will then cover some specific WMI providers that Microsoft has made available in Windows Server 2003; these providers help you monitor certain aspects of Active Directory, such as trusts and replication. In the next chapter, we will cover the WMI DNS provider and how you can manage Microsoft DNS servers with it. To start with, let's look at the Active Directory provider.

The Active Directory provider uses the root\directory\ldap namespace. Within that namespace, every Active Directory schema class and attribute is mapped to corresponding WMI classes or properties. Each abstract class (e.g., top) is mapped to a WMI class with "ds_" prefixed on the name. Each nonabstract class (e.g., structural and auxiliary) is mapped to two classes. One has "ads_" prefixed, and the other has "ds_" prefixed. The "ads_" classes conform to the class hierarchy defined by the subClassOf attribute for each class. The "ds_" classes for nonabstract (e.g., structural) classes are descendants of their cooresponding "ads_" class. Perhaps an example would help illustrate this hierarchy:

```
ds_top
  ads_person
      ads_organizationalperson
          ads_user
              ads_computer
                   ds_computer
```

In this example, we showed the class hierarchy for the Active Directory "computer" object class as it is mapped to WMI. The attribute mappings are more straightforward. Each Active Directory attribute has a corresponding property in WMI with "ds_" prefixed. So the description attribute would map to the ds_description property in WMI. An additional property was added called ADSIPath, which is the ADsPath, and is the key for each Active Directory object in WMI. We highly recommend installing and using the WMI CIM Studio to browse the root\directory\ldap namespace. The organization of classes and objects will become apparent.

We can use the techniques shown so far to query and manipulate Active Directory objects. We can retrieve all the instances of a particular Active Directory class (via InstancesOf) or perform WQL query based on certain criteria. In the following example, we search for all user objects that have a last name equal to "Allen".

```
strComputer = "."

Set objWMI = GetObject("winmgmts:\\" & strComputer & "\root\directory\LDAP")
Set objUsers = objWMI.ExecQuery("SELECT * FROM ds_user where ds_sn = 'Allen' ")

if objUsers.Count = 0 then
   Wscript.Echo "No matching objects found"
else
   for each objUser in objUsers
      WScript.Echo "First Name: " & objUser.ds_givenName
      WScript.Echo "Last Name: " & objUser.ds_sn
      WScript.Echo ""
   next
end if
```

Since WMI is typically used to manage computers, we can leverage Active Directory as a repository of computer objects and perform certain functions on a set of computers that match our criteria. In the next code sample, we do a WQL query for all computers that are running "Windows Server 2003", connect to each one, and print the date each machine was last rebooted.

```
on error resume next

strComputer = "."
```

26.8 Monitoring Trusts

New to Windows Server 2003 is the Trustmon WMI provider. The Trustmon provider allows you to query the list of trusts supported on a domain controller and determine if they are working correctly. The Trustmon provider consists of three classes, but the primary one is the Microsoft_DomainTrustStatus class, which represents each trust the domain controller knows about. The Trustmon provider is contained under the root\MicrosoftActiveDirectory namespace. Note that this namespace is different than for the Active Directory provider, which is contained under root\directory\ldap.

Table 26-6 provides a list of the property methods available to this class.

Table 26-6. Microsoft_DomainTrustStatus properties

Property	**Description**
Flatname	NetBIOS name for the domain.
SID	SID for the domain.
TrustAttributes	Flag indicating special properties of the trust. Can be any combination of the following: • 0x1 (Nontransitive) • 0x2 (Uplevel clients only) • 0x40000 (Tree parent) • 0x80000 (Tree root)
TrustDCName	Name of the domain controller the trust is set up with.
TrustDirection	Integer representing direction of the trust. Valid values include: • 1 (Inbound) • 2 (Outbound) • 3 (Bidirectional)
TrustedDomain	Naming of trusted domain.
TrustIsOK	Boolean indicating whether the trust is functioning properly.

26.9 Monitoring Replication

The WMI Replication provider is another good example of how Microsoft is leveraging WMI to help with monitoring Active Directory. Like the Trustmon provider, the Replication provider is only available with Windows Server 2003 and is contained under the root\MicrosoftActiveDirectory namespace. It provides classes to list the replication partners for a domain controller, view the supported Naming Contexts for a domain controller, and also see the pending replication operations.

As of the time of this writing, Microsoft had not published any documentation on the Replication provider. Most of the information contained in this section was observed by one of the authors and is likely not to be the complete story!

Table 26-8 contains some of the more useful properties for the MSAD_ReplNeighbor class, which represents a replication partner (or neighbor) for a given domain controller.

Table 26-8. Useful MSAD_ReplNeighbor properties

Property	Description
IsDeletedSourceDsa	Boolean indicating whether the source DC has been deleted.
LastSyncResult	Number representing the result of the last sync operation with this neighbor. A value of 0 indicates success.
NamingContextDN	DN of the Naming Context for which the partners replicate.
NumConsecutiveSyncFailures	Number of consecutive sync failures between the two neighbors.
SourceDsaCN	CN of the replication neighbor.
SourceDsaSite	Site the replication neighbor is in.
TimeOfLastSyncAttempt	Time of the last sync attempt.
TimeOfLastSyncSuccess	Time of last successful sync attempt.

There are actually several property methods available other than what is shown in Table 26-8, so in the following example, we will enumerate all the replication neighbors and print out every property available to the MSAD_ReplNeighbor class.

```
 strComputer = "."

Set objWMI = GetObject("winmgmts:\\" & strComputer & _
                       "\root\MicrosoftActiveDirectory")
Set objReplNeighbors = objWMI.ExecQuery("Select * from MSAD_ReplNeighbor")

for each objReplNeighbor in objReplNeighbors

   Wscript.Echo objReplNeighbor.SourceDsaCN & "/" &
```

26.10 Summary

In this chapter we gave a quick introduction into the WMI architecture and the concepts behind it. We then covered some of the tools available for querying and modifying WMI data. Next we went through several examples for querying and manipulating services and the Event Logs. The last part of the chapter covered the WMI hooks into Active Directory, including the WMI providers for Trustmon and Replication monitoring.

In the next chapter we will put our WMI knowledge to use as we work with the WMI DNS Provider. We will use WMI to configure Microsoft DNS server settings programmatically and manipulate zones and resource records.

Chapter 27. Manipulating DNS

DNS is a core technology of Active Directory that cannot be overlooked. While features such as Active Directory Integrated DNS can take a lot of the hassle of managing DNS servers and zones out of your hands, you still have to set up the initial zone configurations. Unfortunately, lack of a good DNS API has always been a big gap for managing a Microsoft DNS server environment. The only way to automate maintenance and management of Microsoft DNS has been by executing Dnscmd commands from within a batch, VBScript, or Perl script. Over time, Microsoft has continued to improve Dnscmd, and as of Windows 2000, it provides just about every option you need to manage DNS server configuration, zones, and resource records using a command line. In Windows Server 2003, it even allows you to manage Application Partitions! Microsoft also provides the DNS MMC snap-in for those that want to manage DNS via a GUI, although it is not very suitable for managing large environments.

Microsoft's answer to the DNS API issue is WMI. As explained in Chapter 26, WMI is Microsoft's API of choice for managing and monitoring systems and services. With the WMI DNS provider, you have complete programmatic control over a Microsoft DNS environment, much as you do with Dnscmd from a command line.

In this chapter, we will cover the WMI DNS provider at length, including the properties and methods available for the primary WMI DNS classes. Several sample scripts will be shown, which will give you a head start on developing scripts to manage your own DNS environment.

27.1 DNS Provider Overview

The WMI DNS provider was first released as part of the Windows 2000 Resource Kit Supplement 1, but unfortunately it was not ready for prime time. That version was buggy, did not include all the documented features, and in several cases behaved differently than what the documentation described. Also, since the DNS provider was included as part of a Resource Kit, it was not fully supported by Microsoft, which means that if you encountered problems, you were largely on your own. That said, much of the functionality you probably need is present in the Windows 2000 version, so it may be suitable. You can download the Windows 2000 DNS provider separately from the Resource Kit via FTP from the following location: ftp://ftp.microsoft.com/reskit/win2000/dnsprov.zip

With Windows Server 2003, the DNS provider is fully functional and supported. It is installed automatically whenever you install the DNS Server service. You can also install it separately as described in the next section. This may be necessary when doing development with the provider on a machine that does not have the DNS Server installed.

For our purposes, all sample code has been tested using the Windows Server 2003 DNS provider.

27.1.1 Installing the DNS Provider

You do not need to manually install the provider if you are installing the DNS Server service on a Windows Server 2003 server because it gets installed with the service.

If you downloaded the DNS provider files for Windows 2000 (*dnsschema.mof* and *dnsprov.dll*), you will first need to copy them to the *%SystemRoot%\System32\wbem* directory. Next, you'll need to compile the DNS managed object format (MOF) file by executing mofcomp *filename* from a command line. With Windows 2000, the DNS MOF file is named *dnsschema.mof*, and with Windows Server 2003 it is called *dnsprov.mof*. The output of the command should look like the following:

```
 C:\WINDOWS\system32\wbem>mofcomp dnsprov.mof
Microsoft (R) 32-bit MOF Compiler Version 5.2.3628.0
Copyright (c) Microsoft Corp. 1997-2001. All rights reserved.
Parsing MOF file: dnsprov.mof
MOF file has been successfully parsed
Storing data in the repository...
Done!
```

The last step is to register the DNS provider DLL by executing regsvr32 dnsprov.dll from a command line. You should see a dialog box with the following:

```
 DllRegisterServer in dnsprov.dll succeeded.
```

At this point you will be able to use the DNS provider from your scripts.

27.1.2 Managing DNS with the DNS Provider

The three main areas of interest when it comes to managing DNS include server configuration, zone management, and creation and deletion of resource records. The DNS provider has several classes to manipulate each of these components. With the MicrosoftDNS_Server class, you can manipulate server configuration settings, start and stop the DNS service, and initiate scavenging. The MicrosoftDNS_Zone class allows you to create, delete, and modify zone configuration. The MicrosoftDNS_ResourceRecord class and child classes provide methods for manipulating the various resource record types. Each of these will be explained in more detail in the next few sections.

Several additional classes are also supported by the DNS provider to manage other aspects of DNS, including the root hints (MicrosoftDNS_RootHints), DNS cache (MicrosoftDNS_Cache), and server statistics (MicrosoftDNS_Statistics). For more information on these classes, including sample scripts in VBScript and Perl, check out the following section in the MSDN Library (http://msdn.microsoft.com/library/): Networking and Directory

27.2 Manipulating DNS Server Configuration

There are close to 50 different settings that can be configured on a Microsoft DNS server. They range from default scavenging and logging settings to settings that customize the DNS server behavior, such as how zone transfers will be sent to secondaries and whether to round-robin multiple A record responses.

The DNS provider is mapped to the root\MicrosoftDNS namespace. A DNS server is represented by an instance of a MicrosoftDNS_Server class, which is derived from the CIM_Service class. Table 27-1 contains all the property methods available in the MicrosoftDNS_Server class.

Table 27-1. MicrosoftDNS_Server class properties

Property name	**Property description**
AddressAnswerLimit	Max number of records to return for address requests (e.g., A records).
AllowUpdate	Determines whether DDNS updates are allowed.
AutoConfigFileZones	Indicates which standard primary zones that are authoritative for the name of the DNS server must be updated when the name server changes.
AutoCacheUpdate	Indicates whether the DNS server will dynamically attempt to update its root hints (also known as cache) file.
BindSecondaries	Determines the format zone transfers (AXFR) will be sent as to non-Microsoft DNS servers.
BootMethod	Determines where the server will read its zone information.
DefaultAgingState	For AD-integrated zones, the default scavenging interval in hours.
DefaultNoRefreshInterval	For AD-integrated zones, the default no-refresh interval in hours.
DefaultRefreshInterval	For AD-integrated zones, the default refresh interval in hours.
DisableAutoReverseZones	Determines whether the server automatically creates reverse zones.
DisjointsNets	Indicates whether the default port binding for a socket used to send queries to remote DNS servers can be overridden.

27.3 Creating and Manipulating Zones

The MicrosoftDNS_Zone class provides a plethora of properties and methods to aid in managing your zones. Even if you are using AD-integrated zones, which help reduce the amount of work it takes to maintain DNS, you will inevitably need to configure settings on a zone or create additional zones. In Table 27-3 and Table 27-4, the list of available properties and methods for the MicrosoftDNS_Zone class are presented.

Table 27-3. MicrosoftDNS_Zone class properties

Property name	**Property description**
AllowUpdate	Flag indicating whether dynamic updates are allowed.
AutoCreated	Flag indicating whether the zone was auto-created.
DataFile	Name of zone file.
DisableWINSRecordReplication	If TRUE, WINS record replication is disabled.
MastersIPAddressesArray	If zone is a secondary, this contains the list of master servers to receive updates from.
Notify	If set to 1, the master server will notify secondaries of zone updates.
NotifyIPAddressesArray	Servers that will be notified when there are updates to the zone.
Paused	Flag indicating whether the zone is paused and therefore not responding to requests.
Reverse	If TRUE, zone is a reverse (in-addr.arpa) zone. If FALSE, zone is a forward zone.
SecondariesIPAddressesArray	Servers allowed to receive zone transfers.
SecureSecondaries	Flag indicating whether zone transfers are allowed only to servers specified in SecondariesIPAddressesArray.
Shutdown	If TRUE, zone has expired (or shutdown).
UseWins	Flag indicating whether zone uses WINS lookups.
ZoneType	Type of zone. It will be either DS Integrated, Primary, or Secondary.

Table 27-4. MicrosoftDNS_Zone class methods

27.4 Creating and Manipulating Resource Records

Resource records are the basic unit of information in DNS. A DNS server's primary job is to respond to queries for resource records. Most people don't realize they are generating resource record queries with nearly every network-based operation they do, including accessing a website, pinging a host, or logging into Active Directory.

Resource records come in many different flavors or types. Each type corresponds to a certain type of name or address lookup. Each record type also has additional information encoded with the record that represents things such as the time to live of the record. The following is a textual example of what a CNAME record looks like:

```
www.mycorp.com.  1800  IN  CNAME  www1.mycorp.com.
```

Or more generically:

```
Owner  TTL  Class  Type  RR-Data
```

Now let's break the record down into its individual parts:

Owner

The owner of the resource record. This field is typically what is specified during a query for the particular type.

TTL

The time to live, or length of time a nonauthoritative DNS server should cache the record. After the TTL expires, a nonauthoritative server should re-query for a authoritative answer.

Class

Resource record classification. In nearly all cases, this will be "IN" for Internet.

Type

Name of the resource record type. Each type has a standard name that is used in zones (e.g., CNAME, A, PTR, SRV).

RR-Data

Resource record specific data. When you perform a query, you are typically looking for the information returned as part of the RR-Data.

The WMI DNS provider fully supports querying and manipulating resource records. In Table 27-5 and Table 27-6, the supported properties and methods are listed for the MicrosoftDNS_ResourceRecord class, which implements a generic interface for resource records.

Table 27-5. MicrosoftDNS_ResourceRecord class properties

Property name	**Property description**
ContainerName	Name of container (e.g., zone name) that holds the RR
DomainName	FQDN of the domain that contains the RR
DnsServerName	FQDN of the server that contains the RR
OwnerName	Owner of the RR
RecordClass	Class of the RR; 1 represents IN
RecordData	Resource record data

27.5 Summary

The WMI DNS provider fills a much-needed gap for programmatic management of a Microsoft DNS environment. In this chapter, we reviewed how to install the DNS provider, including some of the caveats for using it on Windows 2000. We then covered the classes used for managing server configuration along with each of the available server settings. Next, we showed how to create and manipulate zones with the DNS provider. Finally, we covered the various resource record types and their associated WMI classes.

Chapter 28. Getting Started with VB.NET and System.Directory Services

Unless you've been hiding in a cave in recent years, you've undoubtedly heard of Microsoft's latest initiative, called .NET. At a low level, .NET is the basis for a new programming platform, including a completely new set of APIs to manage Microsoft-based products and develop Windows applications. Microsoft even released a new programming language in conjunction with .NET called C# (C-sharp). At a higher level, Microsoft has morphed the concept of .NET to the point where it is hard to define its true boundaries. Here is the definition provided on Microsoft's website: "Microsoft .NET is a set of software technologies designed to connect your world of information, people, systems, and devices."

As far as Active Directory goes, the impact of .NET has been pretty minimal so far. Windows Server 2003 Active Directory was an evolutionary step, not revolutionary. Perhaps the biggest .NET-influenced change is with the new APIs called System.DirectoryServices that were developed for Active Directory. In this chapter, we will discuss the System.DirectoryServices interfaces and cover numerous examples for how they can be used to query and manipulate data in Active Directory. Before getting into that, we first need to talk a bit about the .NET Framework.

28.1 The .NET Framework

The .NET Framework is a new set of interfaces intended to replace the old Win32 and COM APIs. A couple of the major design goals for the .NET Framework were to make programming in a Windows environment much simpler and more consistent. The .NET Framework has two major components: the common language runtime (CLR) and the .NET Framework class library.

The CLR is the sandbox from which all .NET-based code, called managed code, is executed. The CLR is in charge of things such as memory management, security management, thread management, and other code management functions. One of the great benefits of the CLR is that different programming languages can develop code that runs in the CLR and can be used by other programming languages. That means you can develop managed Perl code that can be easily used by a C# application.

The other major component of the .NET Framework is the class library, which is a comprehensive set of object-oriented interfaces that replace the traditional Win32 API. The class library is divided up into namespaces. You can think of a namespace as a grouping of classes, properties, and methods that are targeted for a specific function. For example, the System.Text namespace contains classes for representing strings in ASCII, Unicode, and other character encoding systems. The namespace that is of the most interest to us is the System.DirectoryServices namespace, which contains all the classes necessary to query and manipulate a directory, such as Active Directory, using the .NET Framework.

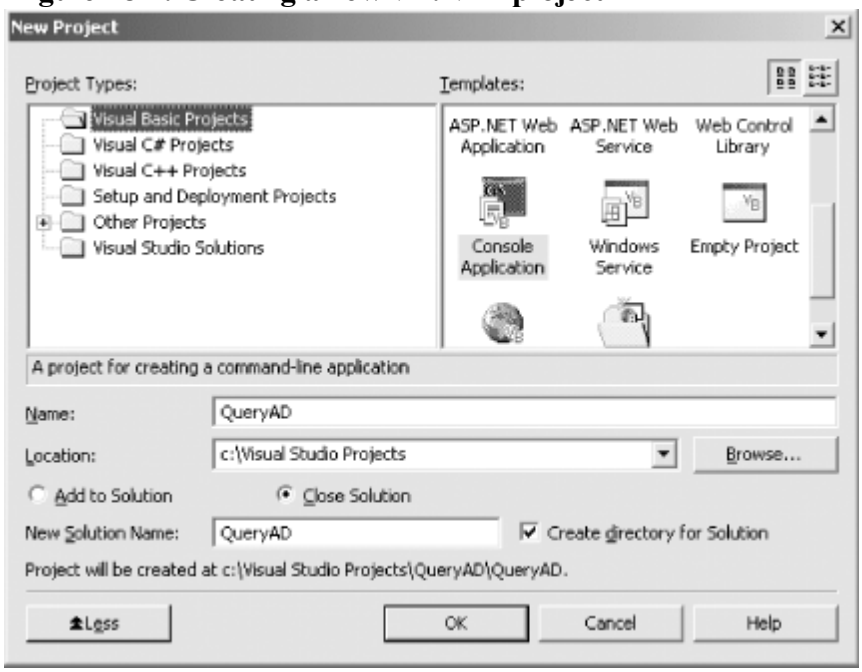
28.2 Using VB.NET

Since the majority of the code we've demonstrated so far in this book has been written in VBScript, you may be wondering why we are going to talk about Visual Basic.NET (VB.NET). Unfortunately, one of the drawbacks with the .NET Framework is that it currently does not provide native support for VBScript. It does support JScript, but since Visual Basic is a much more powerful language than JScript, we will use VB.NET in our examples. It is still unclear what Microsoft's future direction is in regard to providing native support for scripting languages like VBScript in .NET. Until that happens, you should get more familiar with the .NET class library and gain some experience with Visual Basic, which will ultimately increase your capabilities as a programmer. As we mentioned earlier, one of the design goals for the .NET Framework was simplicity. With the .NET Framework class library, Microsoft has made developing Windows-based applications significantly easier. As far as Active Directory goes, it will not take long at all to map your ADSI knowledge to the classes, properties, and methods in the System.DirectoryServices namespace.

To get started using VB.NET, you'll need to get an integrated development environment (IDE) such as Visual Studio.NET (VS.NET), which is available from http://msdn.microsoft.com/vstudio/. Once you have VS.NET, you should download the latest .NET Framework SDK, which is available from http://msdn.microsoft.com/netframework/. Once you have both of those installed, you are ready to start programming with the .NET Framework.

To start a new project in VS.NET, select File ⟶ New ⟶ Project from the menu. At that point you'll see a screen similar to the one in Figure 28-1.

Figure 28-1. Creating a new VB.NET project



Click on Visual Basic Projects and select Console Application from the Templates window. Now you have started a new project and are ready to start writing code in a file called *Module1.vb*, which contains the following code by default:

```
Module Module1
    Sub Main(  )
    End Sub
End Module
```

If you are inexperienced with VB, you can create usable programs simply by adding code to the Main( ) subroutine. Once you become more experienced, you can start creating your own classes, subroutines and functions, and reference them within Main( ).

To start using the System.DirectoryServices classes to query and manipulate Active Directory, you must add a reference to it in your project. From the menu, select Project ⟶ Add Reference, then under Component Name click on System.DirectoryServices. Click the Select button and click OK. Figure 28-2 shows what this window looks like in VS.NET.

28.3 Overview of System.DirectoryServices

The System.DirectoryServices namespace contains several classes, many of which were built on top of ADSI. If you are already familiar with ADSI, the learning curve for the System.DirectoryServices classes should be pretty minimal. Table 28-1 contains the base classes contained within the System.DirectoryServices namespace.

Table 28-1. System.DirectoryServices classes

Class name	Description
DirectoryEntries	Contains the children (child entries) of an entry in Active Directory.
DirectoryEntry	Encapsulates a node or object in the Active Directory hierarchy.
DirectorySearcher	Performs queries against Active Directory.
DirectoryServicesPermission	Allows control of code access security permissions for System.DirectoryServices.
DirectoryServicesPermissionAttribute	Allows declarative System.DirectoryServices permission checks.
DirectoryServicesPermissionEntry	Defines the smallest unit of a code access security permission set for System.DirectoryServices.
DirectoryServicesPermissionEntryCollection	Contains a strongly typed collection of DirectoryServicesPermissionEntry objects.
PropertyCollection	Contains the properties of a DirectoryEntry.
PropertyValueCollection	Contains the values of a DirectoryEntry property.
ResultPropertyCollection	Contains the properties of a SearchResult instance.
ResultPropertyValueCollection	Contains the values of a SearchResult property.
SchemaNameCollection	Contains a list of the schema names that the SchemaFilter property of a DirectoryEntries object can use.
SearchResult	Encapsulates a node in the Active Directory hierarchy that is returned during a search through DirectorySearcher.
SearchResultCollection	Contains the SearchResult instances that the Active Directory hierarchy returned during a DirectorySearcher

28.4 DirectoryEntry Basics

The DirectoryEntry class contains several properties to access the attributes of Active Directory objects. The following code shows how to display the currentTime attribute of the RootDSE:

```
 Dim objRootDSE As New DirectoryEntry("LDAP://RootDSE")
Console.WriteLine(objRootDSE.Properties("currentTime")(0))
```

In the code, once we instantiated the DirectoryEntry object, we can access the currentTime attribute by passing it to the Properties property. The Properties property actually returns a collection of values for the attribute in the form of a PropertyCollection class, which is why we needed to specify an index of 0 to get at a specific value. If the currentTime attribute was multivalued, we could get at the other values by incrementing the index to 1 and so on.

In object-oriented parlance, a property allows you to get or set an attribute of an object. A method typically results in some kind of action being taken on the object.

Now let's look at how to display all of the values for all of the attributes of an object in Active Directory. Again we will use RootDSE as the object we want to display:

```
 Dim objRootDSE As New DirectoryEntry("LDAP://RootDSE")
Dim strAttrName As String
Dim objValue As Object
For Each strAttrName In objRootDSE.Properties.PropertyNames
    For Each objValue In objRootDSE.Properties(strAttrName)
        Console.WriteLine(strAttrName & " : " & objValue.ToString)
    Next objValue
Next strAttrName
```

As you can see, the Properties property, which returns a PropertyCollection, has a PropertyNames property that returns a collection of attribute names for the Active Directory object. We loop over each attribute name and then loop over each value for that attribute to ensure we print out values for all single- and multivalued attributes. The ToString property converts whatever value is stored in the attribute to a printable string.

There are several properties available in the DirectoryEntry class. Table 28-2 contains a list of them.

Table 28-2. DirectoryEntry properties

Property name	**Description**
AuthenticationType	Gets or sets the type of authentication to use when accessing the directory.
Children	Gets a DirectoryEntries class that contains the child objects of this object.
Guid	Gets the GUID for the object (e.g., in Active Directory the objectGUID attribute).
Name	Gets the relative distinguished name of the object.
NativeGuid	Gets the GUID of the object as returned by the provider.
NativeObject	Gets the native ADSI object.

28.5 Searching with DirectorySearcher

We've shown how easy it is to read individual objects from Active Directory with the DirectoryEntry class, so let's now look at how to search Active Directory with the DirectorySearcher class. The DirectorySearcher class works like many other LDAP-based search APIs. Table 28-4 contains all of the DirectorySearcher properties.

Table 28-4. DirectorySearcher properties

Property name	Description
CacheResults	Gets or sets the flag that determines whether results are cached on the client.
ClientTimeout	Gets or sets the time period the client is willing to wait for the server to answer the search.
Filter	Gets or sets the search filter string.
PageSize	Gets or sets the page size for paged searching.
PropertiesToLoad	Gets or sets the attributes to return from a search.
PropertyNamesOnly	Gets or sets the flag indicating to only return attribute names from a search.
ReferralChasing	Gets or sets whether referrals are chased.
SearchRoot	Gets or sets the base from which the search should start.
SearchScope	Gets or sets the scope of the search.
ServerPageTimeLimit	Gets or sets the time the server will wait for an individual page to return from a search.
ServerTimeLimit	Gets or sets the time the server will wait for a search to complete.
SizeLimit	Gets or sets the maximum number of objects that can be returned by a search.
Sort	Gets or sets the attribute that is used when returning sorted search results.

Many of the properties, such as SearchScope, should look familiar. The following code shows how to search for all user objects in the mycorp.com domain.

```
 Dim objSearch As New DirectorySearcher(  )
objSearch.SearchRoot = New DirectoryEntry("LDAP://dc=mycorp,dc=com")
objSearch.Filter = "(&(objectclass=user)(objectcategory=person))"
objSearch.SearchScope = SearchScope.Subtree
```

28.6 Manipulating Objects

Modifying objects with System.DirectoryServices can be done a couple of different ways. To modify an attribute that currently has a value, you can set it using the Properties property. For example, the following code would modify the givenName attribute:

```
objADObject.Properties("givenName")(0) = "Robert"
```

If you want to set an attribute that was previously unset, you must use the Properties.Add method. The following code would set the previously unset sn attribute:

```
objADObject.Properties("sn").Add("Robert")
```

To determine whether an attribute has been set, you can use Properties("*attributename*").Count, which will return the number of values that have been set for the attribute. Just like with ADSI, all modifications are made initially to the local property cache and must committed to the server. With ADSI you would use the IADs::SetInfo( ) method, and with System.DirectoryServices it is called CommitChanges( ), which is available from the DirectoryEntry class.

```
objADObject.CommitChanges(  )
```

Now that we covered how to set an attribute, we can modify the earlier code that printed all the values of an attribute to instead set an attribute. The code in Example 28-2 expects three command line parameters: the first is the ADsPath of the object to modify, the second is the attribute name, and the third is the value to set the attribute to.

Example 28-2. Setting an attribute

```
 Dim strADsPath As String
Dim strAttrName As String
Dim strAttrValue As String
Try
    Dim intArgs As Integer = Environment.GetCommandLineArgs(  ).Length(  )
    If intArgs <> 4 Then
        Throw (New Exception("All parameters are required"))
    Else
        strADsPath = Environment.GetCommandLineArgs(  )(1)
        strAttrName = Environment.GetCommandLineArgs(  )(2)
        strAttrValue = Environment.GetCommandLineArgs(  )(3)
    End If
Catch objExp As Exception
    Console.WriteLine("Error: " & objExp.Message)
    Console.WriteLine("Usage: " & Environment.GetCommandLineArgs(  )(0) & _
                      " ADsPath AttributeName Attribute Value")
    Console.WriteLine(  )
    Return
End Try
Dim objADObject As New DirectoryEntry(  )
Try
    If objADObject.Exists(strADsPath) = False Then
        Throw (New Exception("Object does not exist"))
    End If
Catch objExp As Exception
    Console.WriteLine("Error retrieving object: " & strADsPath)
    Console.WriteLine("Error: " + objExp.Message)
    Return
End Try
Dim strOldValue As String
Try
    objADObject.Path = strADsPath
    If objADObject.Properties(strAttrName).Count > 0 Then
        strOldvalue = objADObject.Properties(strAttrName)(0)
        objADObject.Properties(strAttrName)(0) = strAttrValue
    Else
        objADObject.Properties(strAttrName).Add(strAttrValue)
    End If
    objADObject.CommitChanges(  )
Catch objExp As Exception
    Console.WriteLine("Error setting object: " & strADsPath)
```

28.7 Summary

The .NET initiative is one of the biggest technology shifts at Microsoft since they embraced the Internet in the latter half of the 1990s. Microsoft is using .NET to refocus the company on new technologies such as XML web services and the .NET Framework. The .NET Framework is a completely new way to program in the Windows environment. The Common Language Runtime (CLR) helps applications share code more efficiently and securely. In addition, the .NET Framework class library is a new set of APIs that make the older Win32 APIs look antiquated. The object-oriented approach and better organization of classes make for a much more simplified programming environment.

The impact of .NET on Active Directory is pretty minimal so far. The biggest impact has been with the introduction of the System.DirectoryServices API, which builds on top of ADSI and is straightforward to use. In its current release, VBScript cannot be used natively with the .NET Framework, but due to the simplicity of .NET, using Visual Basic.NET is not much of a leap for experienced VBScript programmers. In this chapter, we covered the two main classes of System.DirectoryServices, the DirectoryEntry class and the DirectorySearcher class. By having a good understanding of these two classes, you'll be well on your way to writing robust Active Directory applications with the .NET Framework.

Colophon

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

The animals on the cover of Active Directory, Second Edition, are a domestic cat (felis silvetris) and her kitten. The domestic cat is a descendant of the African wild cat, which first inhabited the planet almost one million years ago. Other early forerunners of the cat existed as many as 12 million years ago.

The domestic cat is one of the most popular house pets in the world. There are hundreds of breeds of domestic cats, which weigh anywhere from five to thirty pounds, with an average of twelve pounds. The cat is slightly longer than it is tall, with its body typically being longer than its tail. Domestic cats can be any of eighty different colors and patterns. They often live to be fifteen to twenty years old; ten years for a human life is about equal to sixty years for a cat.

The cat's gestation period is approximately two months, and each litter may contain three to seven kittens. Mother cats teach their kittens to eat and to use litter boxes. Kittens ideally should not leave their mother's side until the age of twelve weeks and are considered full-grown at the age of about three years.

Darren Kelly was the production editor and Leanne Soylemez was the copyeditor for Active Directory, Second Edition. Mary Brady, Tatiana Apandi Diaz, Mary Anne Weeks Mayo, and Claire Cloutier provided quality control. Derek Di Matteo and Jamie Peppard provided production support. Reg Aubry wrote the index.

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

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

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

[Team LiB]

[Team LiB]