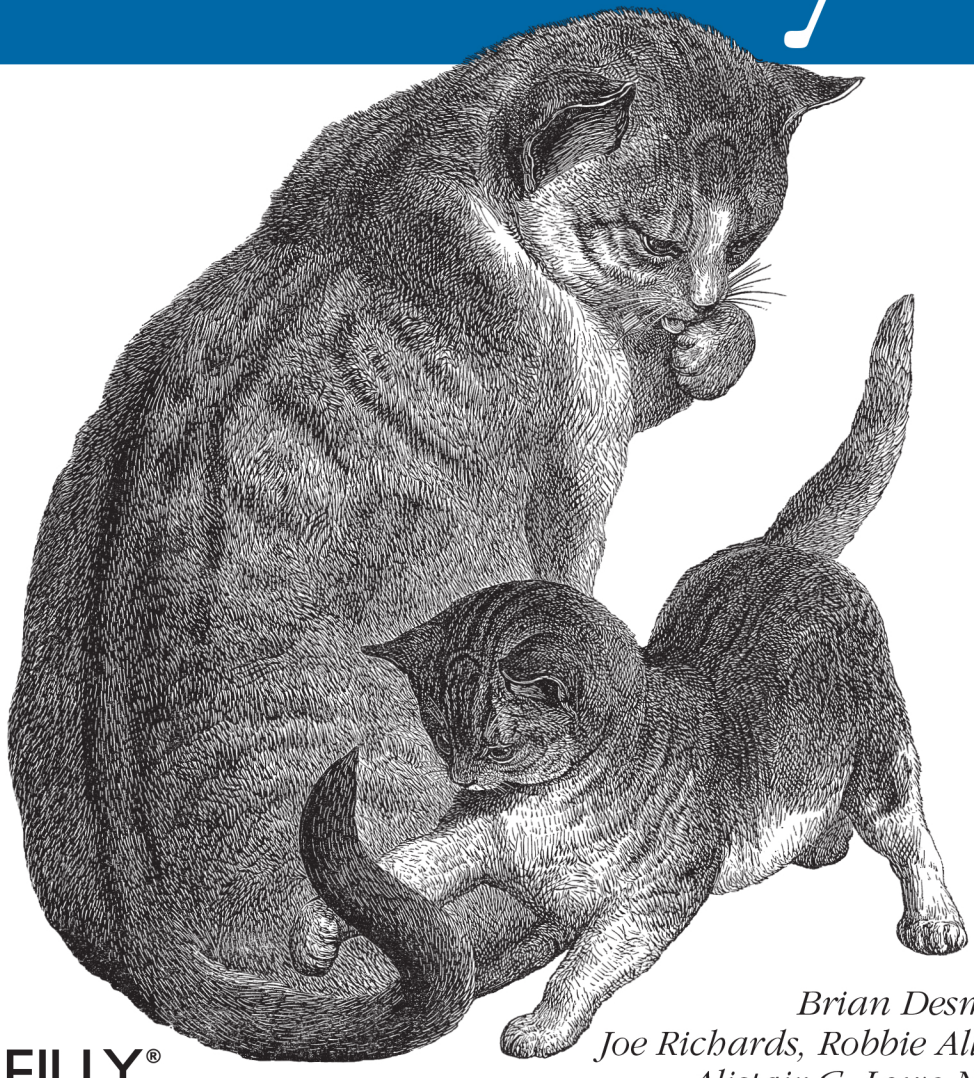
Designing, Deploying, and Running Active Directory

5th Edition
Covers Windows Server 2003 - 2012 and PowerShell

Active Directory



Brian Desmond, Joe Richards, Robbie Allen & Alistair G. Lowe-Norris

O'REILLY®

Active Directory

Organize your network resources by learning how to design, manage, and maintain Active Directory. Updated to cover Windows Server 2012, the fifth edition of this bestselling book gives you a thorough grounding in Microsoft's network directory service by explaining concepts in an easy-to-understand, narrative style.

You'll negotiate a maze of technologies for deploying a scalable and reliable AD infrastructure, with new chapters on management tools, searching the AD database, authentication and security protocols, and Active Directory Federation Services (ADFS). This book provides real-world scenarios that let you apply what you've learned—ideal whether you're a network administrator for a small business or a multinational enterprise.

- Upgrade Active Directory to Windows Server 2012
- Learn the fundamentals, including how AD stores objects
- Use the AD Administrative Center and PowerShell
- Learn AD Federation Services
- Search and gather AD data, using the LDAP query syntax
- Understand how Group Policy functions
- Design a new Active Directory forest
- Examine the Kerberos security protocol
- Get a detailed look at the AD replication process
- Explore AD Lightweight Directory Services

Brian Desmond, Microsoft MVP since 2003, is a consultant focused on Active Directory and identity management for some of the world's largest companies.

US $54.99 CAN $57.99
ISBN: 978-1-449-32002-7

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

Active Directory

Brian Desmond, Joe Richards,
Robbie Allen, and Alistair G. Lowe-Norris

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Active Directory

by Brian Desmond, Joe Richards, Robbie Allen, and Alistair G. Lowe-Norris

Printed in the United States of America.

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

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

Editor: Rachel Roumeliotis
Production Editor: Rachel Steely
Copyeditor: Jasmine Kwityn
Proofreader: Rachel Head
Indexer: Bob Pfahler
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrators: Robert Romano and Rebecca Demarest

April 2013: Fifth Edition

Revision History for the Fifth Edition:

2013-04-10: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449320027* for release details.

ISBN: 978-1-449-32002-7

[LSI]

Table of Contents

Preface

Active Directory is a common repository for information about objects that reside on the network, such as users, groups, computers, printers, 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 each object, which allows you to maintain permissions for who can access and manage the object. Having a single source for this information makes it more accessible and easier to manage; however, accomplishing this requires a significant amount of knowledge on such topics as the Lightweight Directory Access Protocol (LDAP), Kerberos, the Domain Name System (DNS), multimaster 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.

This book is a major update to the very successful fourth edition. All of the existing chapters have been brought up to date through Windows Server 2012, in addition to updates in concepts and approaches to managing Active Directory and script updates. There are five new chapters (Chapter 3, Chapter 7, Chapter 10, Chapter 19, and Chapter 21) to explain features or concepts not covered in previous editions. These chapters include in-depth coverage of management tools, LDAP query syntax, Kerberos, Active Directory Federation Services (ADFS), and more.

This book 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 how to design, manage, and maintain a small, medium, or enterprise Active Directory infrastructure.

We begin in general terms with 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, interaction with DNS, domain controllers, password policies, Kerberos, and LDAP.

Next, 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, auditing, permissions, Dynamic Access Control (DAC), backup and recovery, Active Directory Lightweight Directory Services (AD LDS, formerly ADAM), upgrading Active Directory, and ADFS.

If you're simply looking for in-depth coverage of how to use the Microsoft Management Console (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 not look any further.

Intended Audience

This book is intended for all Active Directory administrators, whether you manage a single server or a global multinational with thousands of servers. Even if you have a previous edition, you will find this fifth edition to be full of updates and corrections and a worthy addition to your "good" bookshelf: the bookshelf next to your PC with the books you really read that are all dog-eared with soda drink spills and pizza grease on them. To get the most out of the book, you will probably find it useful to have a server running Windows Server 2012 available so that you can check out various items as we point them out.

Contents of the Book

Chapter 1, A Brief Introduction
: Reviews the evolution of the Microsoft network operating system (NOS)and some of the major features and benefits of Active Directory.

Chapter 2, Active Directory Fundamentals
: 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, Active Directory Management Tools
: Demonstrates how to use the various MMC snap-ins and management tools that are commonly used by Active Directory administrators.

Chapter 4, Naming Contexts and Application Partitions
: Reviews the predefined naming contexts within Active Directory, what is contained within each, and the purpose of application partitions.

Chapter 5, Active Directory Schema
: Describes how the blueprint for each object and each object's attributes are stored in Active Directory.

Chapter 6, Site Topology and Active Directory Replication
: Details how the actual replication process for data takes place between domain controllers.

Chapter 7, Searching Active Directory
: Explains the LDAP query syntax used for gathering data from Active Directory.

Chapter 8, Active Directory and DNS
: Describes the importance of the Domain Name System and what it is used for within Active Directory.

Chapter 9, Domain Controllers
: Describes the deployment and operation of writable and read-only domain controllers (RODCs) as well as the impacts of hardware virtualization on Active Directory.

Chapter 10, Authentication and Security Protocols
: Describes the Kerberos security protocol that is fundamental to Active Directory, as well as managed service accounts.

Chapter 11, Group Policy Primer
: Provides a detailed introduction to the capabilities of group policy objects and how to manage them.

Chapter 12, Fine-Grained Password Policies
: Gives comprehensive coverage of how to design, implement, and manage fine-grained password policies.

Chapter 13, Designing the Active Directory Structure
: 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 unit(s).

Chapter 14, Creating a Site Topology
: 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 15, Planning for Group Policy
: 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 16, Active Directory Security: Permissions and Auditing
: Describes how you can design effective security for all areas of your Active Directory infrastructure, both in terms of access to objects and their properties; includes

information on how to design effective security access logging in any areas you choose. This chapter also covers Dynamic Access Control.

Chapter 17, Designing and Implementing Schema Extensions
: Covers procedures for extending the classes and attributes in the Active Directory schema.

Chapter 18, Backup, Recovery, and Maintenance
: Describes how you can back up and restore Active Directory, from the entire directory down to the object level.

Chapter 19, Upgrading Active Directory
: Discusses the features introduced in each version of Active Directory, followed by an outline of how you can upgrade your existing Active Directory infrastructure to Windows Server 2012.

Chapter 20, Active Directory Lightweight Directory Services
: Introduces Active Directory Lightweight Directory Services.

Chapter 21, Active Directory Federation Services
: Introduces Active Directory Federation Services.

Appendix A
: Starts off by providing some background information on the .NET Framework and then dives into several examples using the `System.DirectoryServices` namespaces with VB.NET.

Conventions Used in This Book

The following typographical conventions are used in this book:

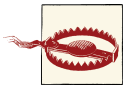
`Constant width`
: Indicates command-line input, computer output, registry keys and values, objects, methods, namespaces, and code examples.

`Constant width italic`
: Indicates text that should be replaced with user-supplied values.

`Constant width bold`
: Indicates user input.

Italic
: Introduces new terms and indicates URLs, commands, command-line utilities and switches, 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.

Using Code Examples

This book is here to help you get your job done. In general, if this book includes code examples, you may use the code in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Active Directory* by Brian Desmond, Joe Richards, Robbie Allen, and Alistair G. Lowe-Norris (O'Reilly). Copyright 2013 Brian Desmond, Joe Richards, Robbie Allen, and Alistair Lowe-Norris, 978-1-449-32002-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT

Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreil.ly/Active_Dir_5E*.

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

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

For the Fourth and Fifth Editions (Brian)

I wouldn't be here if it weren't for the fine folks at O'Reilly who decided to entrust this project to me. Special thanks to editors Rachel Roumeliotis and Laurel Ruma, who made this a very smooth-running adventure. Joe, Robbie, and Alistair have of course provided an excellent foundation, which made this project so much easier. I would not have been able to get this done in the time I did without their hard work.

There are numerous individuals whose contributions to the depth and accuracy of the content in these latest editions are irreplaceable. Without their help, this book would not be what it is:

- .NET expert Joe Kaplan contributed the fine content in this book on this important topic.
- Technical reviewers Joe Richards, Mark Parris, Mark Morowczynski, Michael B. Smith, and Guido Grillenmeier, thank you for the comments, corrections, and invaluable feedback. Mark Morowczynski and Guido Grillenmeier, thank you for voluntarily taking the time out of your days and vacations to provide your expertise.
- Special thanks to Eric Kotz. Your feedback from the perspective of an Active Directory beginner brought clarity to the chapters you reviewed.
- Thank you to Microsoft experts Mark Morowczynski, Dean Wells, James McColl, Siddharth Bhai, Dmitri Gavrilov, Eric Fleischman, and Stephanie Cheung for your help with the details that made this book what it is!
- Darren Mar-Elia (C-GPO), your feedback on the Group Policy chapters was instrumental.
- Dean Wells, your crucial assistance in decrypting English phraseology is priceless, and of course thanks for your help in consistently transforming complex technical content to plain English.
- Susan Bradley, Small Business Server Diva, your contributions were critical.
- Jorge de Almeida Pinto (Princess), thank you for the last-minute contributions to our list of new Active Directory features in Windows Server 2008.

John Tanner, thanks for all your help behind the scenes, making the Fourth Edition successful. Matt Wagner at Fresh Books, your assistance and expertise in handling the business end of this project were key.

Patrick Sheren and Scott Weyandt, thank you for the opportunity you gave me. I would not be where I am today if it weren't for the years we spent working together. And yes, you too, Kurt.

To the special people in my life who are always trying to get me to explain what I do all day, you have provided the impetus for this project. Thank you for putting up with the hours I spent in my home office working on it.

To my readers, I had a lot of fun on this project, and I hope you have as much fun reading this book as I had writing it.

For the Third Edition (Joe)

I want to thank Robbie Allen for my introduction into the world of book writing and for putting up with my often-grumpy responses to silly issues we encountered on this project. Truly, I wouldn't have worked on this book had it not been for Robbie; if I did not say it before, I am happy I had the opportunity to have this experience—thank you.

Thanks to Alistair for the first edition. I recall being involved with the decision to migrate a company of 200k+ users to Windows 2000 and realizing that I knew nothing about Active Directory (AD) other than it was supposed to be "super-cool" and fixed everything that was broken in NT. "The Cat Book," the *only* book on AD around at the time, prepared me with the essential concepts and ideas to get started. After five years, I am happy to be able to give back some of what I have learned to that very same book.

Thanks to the folks who had the onerous task of finding the mistakes. I was lucky to have very knowledgeable reviewers who spent a lot of time reading every word (old and new) and bluntly telling me the issues. To Hunter Colman and Stuart Fuller: you guys were afraid you wouldn't add value. You were completely wrong; you added a lot of value. To Lee Flight: thanks for reviewing another edition of this book; your comments were invaluable. To Laura Hunter: I will never look at a comma the same way again; you helped the structure and flow immensely. To Ulf B. Simon-Weidner: your comments and ideas were a great help. Finally, thanks to Dean Wells, a great source of information, fear, and humorous English phrases. Dean couldn't review everything but he happily helped me out when I asked. He spent at least 90 minutes on the phone one night just discussing changes that needed to be made to a few pages of Chapter 5. All of these guys (and the gal) are extremely knowledgeable, opinionated, and professional. It was an honor having them tell me what was screwed up. Thanks to my friend Vern Rottmann for being an "unofficial" reviewer and running interference for me when I worked with him.

Thanks to the Microsoft Directory Service Developers: because of you, we have a "super-cool" DS. P.S.: AD/AM rocks. Thanks to Dmitri Gavrilov for going above and beyond by responding to my unsolicited emails. Thanks to Stuart Kwan (of the Ottawa Kwan Clan) for being one of the most insanely energetic speakers and, at the same time, actually listening to what we thought was wrong and working to get corrections. I am thrilled that someday I will be able to run DCs without IE loaded. May your energizer battery never run out of juice. Thanks to Brett Shirley for telling me to correct stuff in Chapter 13 and writing the most brilliant parts of REPADMIN and being a killer JET Blue (ESE) dev. Thanks to Eric Fleischman for answering all the random AD questions from myself as well as everyone else at all hours of the day and night. Your answers, comments, thoughts, and insight into the actual questions themselves are all greatly appreciated.

Thanks to the *http://activedir.org* listserv crowd. Hands down, that list is the best Active Directory (and often Exchange) resource outside of Microsoft. It has helped me a lot.

And last but not least, thanks to my family, great people I love without bound.

For the Second Edition (Robbie)

I would like to thank the people at O'Reilly for giving me the opportunity to work on this book. Special thanks goes to Robert Denn, who was a great editor to work with.

I would like to thank Alistair Lowe-Norris for providing such a solid foundation in the first edition. While there was a lot of new material to include, much of the information in the first edition was still pertinent and useful. He deserves a lot of credit since the first edition was done before Windows 2000 had even been released to the public, and there was virtually no information on Active Directory available.

Thanks to Alistair, Mitch Tulloch, and Paul Turcotte for providing very insightful feedback during the review process. Their comments rounded out the rough edges in the book.

And no acknowledgments section would be complete without recognition to my significant other, Janet. She was supportive during the many late nights and weekends I spent writing. I appreciate everything she does for me.

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, his 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 UK Professional Developers Conference in 1997; Julius Davies for various discussions and, among other things, telling me how the auditing and permissions aspects of Active Directory had all changed just after I finished the chapter; and 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: especially my editor Robert Denn, for his encouragement, patience, and keen desire to get this book crafted properly.

CHAPTER 1
A Brief Introduction

Active Directory (AD) is Microsoft's network operating system (NOS). Originally built on top of Windows 2000, AD has evolved over the course of more than a decade through multiple major Windows releases. This book covers Active Directory through the Windows Server 2012 release.

Active Directory 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 enterprise, 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.

This book is a comprehensive introduction to Active Directory with a broad scope. In the next few chapters, we cover many of the basic concepts of Active Directory to give you a good grounding in some of the fundamentals that every administrator should understand. Then 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.

Before moving on to some of the basic components within Active Directory, though, we will take a moment to review how Microsoft came to the point of implementing a Lightweight Directory Access Protocol (LDAP)-based directory service to support its NOS environment.

Evolution of the Microsoft NOS

Network operating system, or "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 by administrators and accessible to end users. Typically, a NOS environment is comprised of one or more servers that provide NOS services, such as authentication, authorization, 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 of the OS/2 operating system. The NT NOS slowly evolved over the next eight years until Active Directory was first released in beta form in 1997.

Under Windows NT, the "domain" concept was introduced, providing a way to group resources based on administrative and security boundaries. NT domains were 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 the need to rearchitect its NOS model into something that would be much more scalable and flexible. It looked to LDAP-based directory services as a possible solution.

A Brief History of Directories

In general 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). Although each of these systems has 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 that could utilize it. One reason is that X.500 is based on the Open System Interconnection (OSI) 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 many features most clients never needed. This prevented large-scale adoption. It was 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 Request for Comments (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 them 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 overhead resulting from 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, LDAPv3, 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, IBM, the OpenLDAP Foundation, and Microsoft have developed LDAP-based directory servers. Most recently, RFC 3377 was released, which lists all of the major LDAP RFCs. For a Microsoft whitepaper on its LDAPv3 implementation and conformance, refer to this website.

Summary

Now that we've given you a brief overview of the origins of Active Directory, we'll leave you to read ahead and learn more about Active Directory. Throughout the rest of this book, we will bring you up to speed with what you need to know to successfully support Active Directory as well as to design an effective Active Directory implementation.

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 feature of Active Directory before embarking on a design, or your design may leave out a critical element.

How Objects Are Stored and Identified

Data stored within Active Directory is presented to the user 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*. Non-container objects are 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. As the name implies, however, a leaf node may not contain any other objects.

Although the data in Active Directory is presented hierarchically, it is actually stored in flat database rows and columns. The directory information tree (DIT) file is an Extensible Storage Engine (ESE) database file. This answers the question "Does Active Directory use JET or ESE database technology?" ESE *is* a JET technology.

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.

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.

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*.

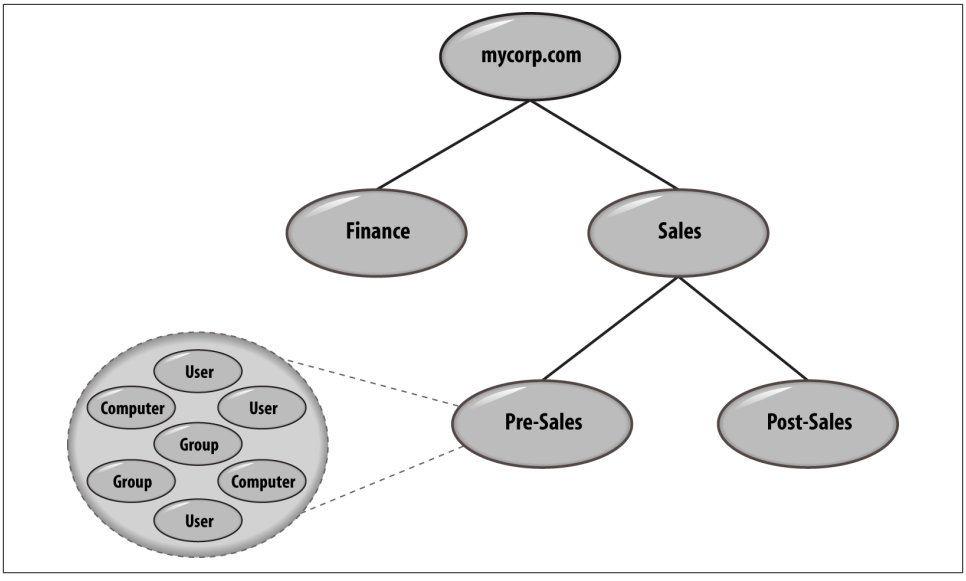


Figure 2-1. A hierarchy of objects

The most common type of container you will create in Active Directory is an organizational unit (OU), but there are others as well, such as the type 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.

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 the Microsoft implementation of the universally unique identifier (UUID) concept from Digital Equipment Corporation. UUIDs/GUIDs are commonly misunderstood to be guaranteed to be unique. This is not the case; the number is just statistically improbable to be duplicated before the year 3400 AD. In the documentation for the GUID creation API function, Microsoft says, "To a very high degree of certainty, this function returns

a unique value." The object's GUID stays with the object until it is deleted, regardless of whether it is renamed or moved within the directory information tree (DIT). The object's GUID will also be preserved if you move an object between domains within a multidomain forest.

A cross-forest move of a security principal using a tool such as the Microsoft Active Directory Migration Tool (ADMT) will not preserve the object's GUID.

Although an object's GUID is 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 a *distinguished name* (DN), is more commonly used.

Distinguished names

Hierarchical paths in Active Directory are known as distinguished names and can be used to uniquely reference an object. Distinguished names are defined in the LDAP standard as a means of referring to any object in the directory.

Distinguished names 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:

```
dc=mycorp,dc=com
```

In the previous distinguished name, you represent the domain root, *mycorp.com*, by separating each part with a comma and prefixing each part with the letters "dc". If the domain had been called *mydomain.mycorp.com*, the distinguished name of the root would have looked like this:

```
dc=mydomain,dc=mycorp,dc=com
```

`dc` stands for *domain component* and is used to specify domain or application partition objects. Application partitions are covered in Chapter 4.

A *relative distinguished name* (RDN) is the name used to uniquely reference an object within its parent container in the directory. For example, this is the DN for the default Administrator account in the Users container in the *mycorp.com* domain:

```
cn=Administrator,cn=Users,dc=mycorp,dc=com
```

This is the RDN of the user:

```
cn=Administrator
```

RDNs must always be unique within the container in which they exist. It is permissible to have two objects with `cn=Administrator` in the directory; however, they must be located inside different parent containers. There cannot be two objects with an RDN of `cn=Administrator` in the Users container.

DNs are made up of names and prefixes separated by the equals sign (=). Another prefix that will become very familiar to you is `ou`, which stands for organizational unit. Here is an example:

```
cn=Keith Cooper,ou=Northlight IT Ltd,dc=mycorp,dc=com
```

All RDNs use a prefix to indicate the class of the object that is being referred to. Any object class that does not have a specific letter code uses the default of `cn`, which stands for common name. Table 2-1 provides a complete list of the most common attribute types amongst directory server implementations. The list is from RFC 2253, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names," and the full text can be found at *http://www.ietf.org/rfc/rfc2253.txt*.

Table 2-1. Attribute types from RFC 2253

Key	Attribute
CN	Common name
L	Locality name
ST	State or province name
O	Organization name
OU	Organizational unit name
C	Country name
STREET	Street address
DC	Domain component
UID	User ID

Active Directory supports using `CN`, `L`, `O`, `OU`, `C`, and `DC`. `CN` or `OU` is used in the majority of cases.

Examples

Let's take a look at Figure 2-1 again. If all the containers were organizational units, the distinguished names for Pre-Sales and Post-Sales would be as follows:

```
ou=Pre-Sales,ou=Sales,dc=mycorp,dc=com
ou=Post-Sales,ou=Sales,dc=mycorp,dc=com
```

And if you wanted to specify a user named Richard Lang, a group called My Group, and a computer called PC1 in the Pre-Sales OU, you would use the following:

```
cn=Richard Lang,ou=Pre-Sales,ou=Sales,dc=mycorp,dc=com
cn=My Group,ou=Pre-Sales,ou=Sales,dc=mycorp,dc=com
cn=PC1,ou=Pre-Sales,ou=Sales,dc=mycorp,dc=com
```

Building Blocks

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

Domains and Domain Trees

Active Directory's logical structure is built around the concept of domains. Domains were 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 and authorizes any access to resources via accounts in the domain or trusts with other domains
- 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. It is not possible to host multiple domains on a single DC. For example, Mycorp has already been allocated a DNS domain name for its company called *mycorp.com*, so it decides that the first Active Directory domain that it is going to build is to be named *mycorp.com*. However, this is only the first domain in a series that may need 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. If Mycorp were to add domains called Europe, Asia, and Americas, then the names would be *europe.mycorp.com*, *asia.mycorp.com*, and *americas.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 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.

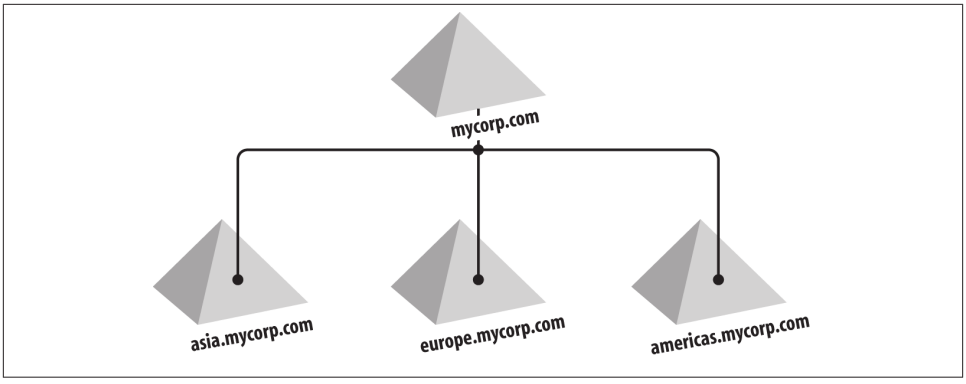


Figure 2-2. The mycorp.com domain tree

Trees ease management and access to resources, as all the domains in a domain tree trust one another implicitly with *transitive trusts*. In a transitive trust, if Domain A trusts Domain B and Domain B trusts Domain C, this implies that Domain A trusts Domain C as well. This is illustrated in Figure 2-3. Put much more simply, the administrator of *asia.mycorp.com* can allow any user in the tree access to any of the resources in the Asia domain that the administrator wishes. The user accessing the resource does not have to be in the same domain.

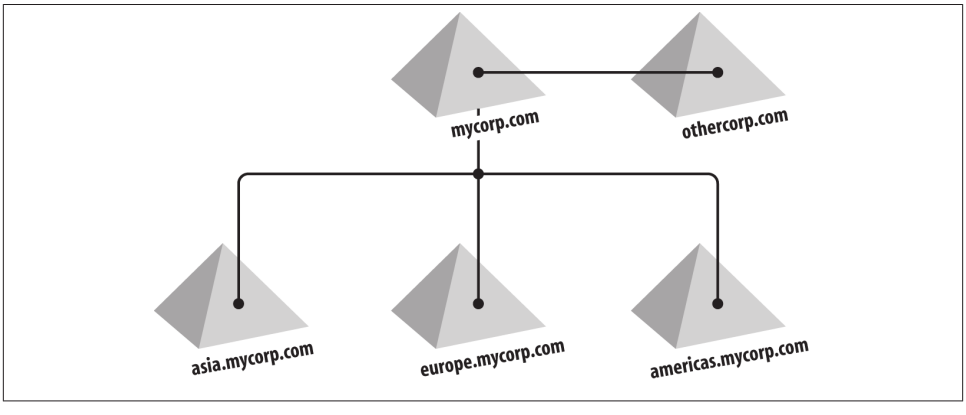


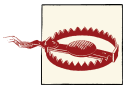
Figure 2-3. The mycorp.com domain tree

Trust relationships do not compromise security; they are just setting up the potential to allow access to resources. Actual access permissions still have to be granted by administrators. This is why you should avoid granting access to Everyone or Authenticated Users on resources. Once a trust is established, everyone in the trusted domain will be able to access those resources as well.

Forests

Now that you understand what a domain tree is, we can move on to the next piece of the Active Directory structure, the forest. Where a domain tree was a collection of domains, a forest is a collection of one or more domain trees. These domain trees share a common Schema and Configuration container, and the trees as a whole are connected together through transitive trusts. As soon as you create a single domain, you have a forest. If you add any domains to the initial domain tree or add new domain trees, you still have one forest.

A forest is named after the first domain that is created, also known as the *forest root domain*. The forest root domain is important because it has special properties.



In Active Directory, you can never remove the forest root domain. If you try to do so, the forest is irretrievably destroyed. Under Windows Server 2003 and newer Active Directories, you can rename the forest root domain, but you cannot change its status as the forest root domain or make a different domain the root.

As we continue with Mycorp, we find that it has a subsidiary business called Othercorp. The DNS domain name allocated to and used by Othercorp is *othercorp.com*. In Othercorp's case, all you would need to do is create the root of the *othercorp.com* tree as a member of the existing forest; *othercorp.com* and *mycorp.com* can then exist together and share resources, as shown in Figure 2-4. The forest containing the *mycorp.com* and *othercorp.com* domain trees is known as the *mycorp.com* forest, in which *mycorp.com* is the forest root domain.

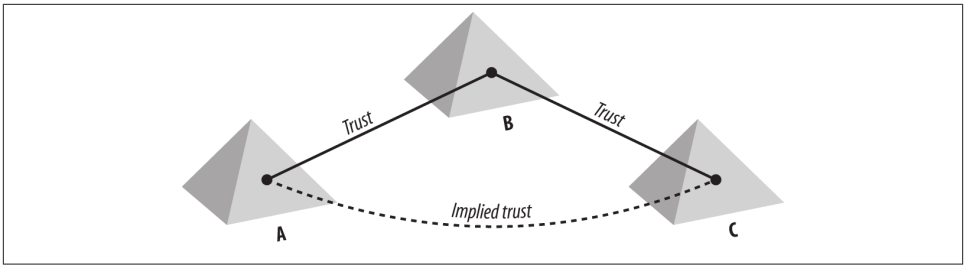
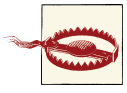


Figure 2-4. Transitive trusts illustrated



While multiple domain trees in a forest can be configured, you should seriously consider all the implications of such a configuration before implementation. It can be confusing for troubleshooting efforts when you are working on an issue in the domain *othercorp.com*, but the configuration information for the forest is maintained in the partition `cn=configuration,dc=mycorp,dc=com`. This is especially true when bringing in outside resources not familiar with the implementation.

While legitimate reasons exist to create multitree forests, we recommend that you endeavor to simplify your Active Directory design as much as possible and limit yourself to one domain tree and as few domains as possible.

Best practice for new forest designs is almost always a single-domain forest. We have outlined the various multidomain options when designing a forest here, but, we do not generally recommend them (including empty roots).

Individual companies often implement their own forests. If Othercorp elected to deploy its Active Directory as a separate forest, you would need to employ a forest trust between Mycorp and Othercorp to provide seamless resource access between the two companies.

A forest trust allows an administrator to create a single transitive one-way or two-way trust between two forest root domains. This trust allows all the domains in one forest to trust all the domains in another forest, and vice versa.

If you have business units that are independent and, in fact, wish to be isolated from each other, then you must not combine them in a single forest. If you simply give each business unit its own domain, these business units are given the impression that they are autonomous and isolated from each other. However, in Active Directory, this level of autonomy and isolation can be achieved only through separate forests. This is also the case if you need to comply with regulatory or legal isolation requirements. Finally, some organizations choose to deploy Microsoft Exchange in a separate resource forest in order to handle separate administrative structures and requirements.

Organizational Units

Having covered the large-scale (domains, trees, and forests) view of Active Directory, we'll now talk about the small scale. When you look inside an Active Directory domain, you will see a hierarchical structure of objects. This hierarchy is made up of objects that can act as containers and objects that cannot. The primary type of container that you will create to house objects is called an organizational unit (OU). Another type of container, which is actually called a *container*, can also be used to store a hierarchy of objects and containers.

Although both can contain huge hierarchies of containers and objects, an organizational unit can have group policies applied to it. (For more information on Group Policy, see Chapter 11.) For this reason, OUs are often used almost exclusively for building object hierarchies within a domain.

Let's illustrate this with an example. Imagine that you are the administrator of the *asia.mycorp.com* domain from Figure 2-2. You have 500 users and 500 computer accounts in the domain. Most of the day-to-day account and machine management is very simple, but the manufacturing section is currently undergoing restructuring and an extensive recruitment program; people keep being transferred in or hired from the outside. You would like to be able to give this group limited autonomy over user objects by allowing one of the senior administrators to manage its own section of the tree. Complete segregation of security is not needed, and the manufacturing tree isn't large enough to justify creating another domain to manage along with the associated domain controllers. You can instead create an organizational unit in your hierarchy called Manufacturing. You then give the senior engineer authority over that organizational unit to create and delete accounts, change passwords, and create other organizational units within the Manufacturing OU. Obviously, the permissions that the senior engineer would be given would be properly tailored so that he had control over only that organizational unit and not the *asia.mycorp.com* domain tree as a whole. You could do this manually or delegate control using the Delegation of Control Wizard, discussed in more depth in Chapter 16.

When you install an Active Directory domain, a number of default containers and organizational units are created automatically, including the Users and Computers containers and the Domain Controllers OU. If you try to create a new container, you will find that there is no option to do so from within the Active Directory Users and Computers (ADUC) MMC snap-in. This also applies to Organization, Locality, and Country container objects. This is intentional; in almost all cases, you would want to create an organizational unit instead of a container. It is possible to create the other types of containers from within scripts and other LDAP tools, but generally it is not necessary. So, throughout this book, whenever we advocate creating hierarchies within domains, we always recommend that you use organizational units. After all, an organizational

unit is just a superset of a container. There is nothing a container can do that an organizational unit cannot.

The Global Catalog

The Global Catalog (GC) is a very important part of Active Directory because it is used to perform forest-wide searches. As its name implies, the Global Catalog is a catalog of all objects in a forest that contains a subset of attributes for each object. The GC can be accessed via LDAP over port 3268 or LDAP/SSL over port 3269. The Global Catalog is read-only and cannot be updated directly.

In multidomain forests, typically you first need to perform a query against the GC to locate the objects of interest. Then you can perform a more directed query against a domain controller for the domain the object is in if you want to access all the attributes available on the object.

The attributes that are available in the Global Catalog are members of the *partial attribute set* (PAS). You can add and remove attributes to and from the PAS by using tools such as the Active Directory Schema snap-in or by modifying the `attributeSchema` object for the attribute directly in the schema.

Under Windows 2000, adding an attribute to the PAS caused all global catalogs in a forest to resynchronize the entire contents of the GC. This could have major replication and network traffic implications. Fortunately, this issue was resolved with Windows Server 2003; a GC resynchronization no longer happens after a PAS addition.

Flexible Single Master Operator (FSMO) Roles

Even though Active Directory is a multimaster directory, there are some situations in which there should only be a single domain controller that can perform certain functions. In these cases, Active Directory nominates one server to act as the master for those functions. There are five such functions that need to take place on one server only. The server that is the master for a particular function or role is known as the *Flexible Single Master Operator* (FSMO, pronounced "fizmo") role owner.

Of the five roles, three exist for every domain, and two apply to the entire forest. If there are four domains in your forest, there will be 14 FSMO roles:

- 2 single forest-wide FSMOs
- 4 sets of 3 domain-wide FSMOs

The number of different role owners can vary greatly depending on whether you have domain controllers serving multiple roles, as is often the case.

The different FSMO roles are the following:

Schema master (forest-wide)
: The schema master role owner is the domain controller that is allowed to make updates to the schema. No other server can process changes to the schema. If you attempt to update the schema on a DC that doesn't hold the schema master FSMO, the DC will return a referral to the schema master role holder. The default schema master role owner is the first server to be promoted to a domain controller in the forest.

Domain naming master (forest-wide)
: The domain naming master role owner is the server that controls changes to the forest-wide namespace. This server adds and removes domains and is required to rename or move domains within a forest, as well as to authorize the creation of application partitions and the addition/removal of their replicas. Like the schema master, this role owner defaults to the first DC you promote in a forest.

It is a common misunderstanding that the schema and domain naming masters cannot be hosted outside of the root domain. Any domain controller in the forest (from any domain) can host the schema and domain naming master FSMO roles. In general, we recommend that these FSMOs be kept on a domain controller in the forest root unless you have a reason to place them elsewhere.

PDC emulator (domain-wide)
: For backward-compatibility purposes, one Active Directory DC has to act as the Windows NT *primary domain controller* (PDC). This server acts as the Windows NT master browser, and it also acts as the PDC for down-level clients. Even though the PDC has very important legacy functions, don't be fooled into thinking that it is no longer important once you have removed all older clients.

  The PDC emulator also has other important functions: for one, it attempts to maintain the latest password for any account. This is enforced by having the other DCs immediately forward any account password changes directly to the PDC. The significance of this feature is in helping to support PDC chaining functions. PDC chaining occurs when an account attempts to authenticate and the local DC doesn't think the password provided is correct. The local DC will then "chain" the authentication attempt to the PDC to see if the PDC thinks the password is okay.

PDC chaining and the matching forwarding of the passwords to the PDC across Active Directory site boundaries can be disabled by setting the `AvoidPdcOnWan` registry value to `1`. This is found in the registry key `HKLM\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters`. If you suspect that PDC chaining isn't working, make sure this registry value isn't configured. You can find more information about this registry setting at this link.

The PDC is also the target server of most Group Policy management tools. This is done to lessen the possibility of the same policy being modified in different ways by different administrators on different DCs at the same time. One other function of the PDC is that the PDC in each domain is the primary time source for the domain, and the PDC of the forest root domain is the primary time source for the entire forest. The *AdminSDHolder* process described in Chapter 16 also runs on the PDC emulator. Finally, the PDC emulator authorizes domain controller cloning operations. For more information about domain controller cloning, see Chapter 9.

RID master (domain-wide)

A *relative identifier* (RID) master exists per domain. Every security principal in a domain has a *security identifier* (SID) that is comprised of several components, including a RID. The system uses the SID to uniquely identify that object for security permissions. In a way, this is similar to the GUID that every object has, but the SID is given only to security-enabled objects and is used only for security verification purposes. For more information about SIDs, see the sidebar "What's in a Security Identifier (SID)?" on page 17. While you may log on or authenticate using the Security Accounts Manager (SAM) account name or universal principal name (UPN) to reference an object, the system always references you for authorization functions by the SID.

In a domain, the SIDs must be unique across the entire domain. As each DC can create security-enabled objects, some mechanism has to exist so that two identical SIDs are never created. To keep conflicts from occurring, the RID master maintains a large pool of unique RID values. When a DC is added to the network, it is allocated a subset of 500 values from the RID pool for its own use. Whenever a DC needs to create a SID, it takes the next available value from its own RID pool to create the SID with a unique value.

In this way, the RID master makes sure that all SIDs in a domain use unique RID values. When a DC's RID pool drops to 50% of the available pool size, the DC contacts the RID master for another set of RID values. The threshold is not set to 0 to ensure that the RID master can be unavailable to other DCs for a brief time without immediately impacting object creations. The RID master itself is in charge of generating and maintaining a pool of unique values across the entire domain.

RID pool size can be configured by setting the `RID Block Size` value in the registry key `HKLM\SYSTEM\CurrentControlSet\Services\NTDS\RID` values on the RID master FSMO role holder. A common scenario where you might use this registry setting is if you have a distributed environment where there can be prolonged connectivity issues between domain controllers and the RID master.

If you decide to use this registry setting, it is a recommended practice to set this value on *any* domain controller that could become the RID master, so you do not have any inconsistencies in RID pool sizes after a RID master FSMO transfer.

What's in a Security Identifier (SID)?

Many Windows administrators know what a SID is: a unique, variable-length identifier used to identify a trustee or security principal. However, few understand what components comprise a SID. A little bit of time spent understanding how SIDs are composed can possibly help an administrator understand the underpinnings of Windows security.

A Windows SID is generally composed of 2 fixed fields and up to 15 additional fields, all separated by dashes like so:

```
S-v-id-s1-s2-s3-s4-s5-s6-s7-s8-s9-s10-s11-s12-s13-s14-s15
```

The first fixed field (`v`) describes the version of the SID structure. Microsoft has never changed this, so it is always 1.

The second fixed field (`id`) is called the *identifier authority*. In Windows domains and Windows computers, it uniquely identifies the authority involved, such as NULL (0), World (1), Local (2), NT Authority (5), etc.

The next 15 fields (`s1`–`s15`) are not required for every SID, and in fact, most SIDs only have a few of these fields populated. These additional fields are called *sub-authorities* and help uniquely identify the object being referenced. The last sub-authority on most SIDs is generally called the RID. This is the value that a domain or computer increments to create unique SIDs.

With that information, you can now look at a SID such as `S-1-5-10` and determine that it is a version 1 SID issued by the NT Authority. This SID is special and is called a *well-known SID*, representing *NT Authority\Self*. Another well known SID is `S-1-1-0`, which is a version 1 World SID; it represents *Everyone*.

There are several other well known SIDs with various values. They are easily identifiable because they don't fit the format of normal computer and domain SIDs. These normal SIDs usually look like this: **`S-1-5-21`**-*xxx-yyy-zzz*-**`r`**, where the

values for *xxx*, *yyy*, and *zzz* are randomly generated when the computer or domain is created. The RID value `r` could either be a consecutive number issued by the RID generation routine or a well known RID assigned to certain security principals that exist in every domain. An example of a well known RID is 500, which translates to the built-in administrator account.

Infrastructure master (domain-wide)

The infrastructure master is used to maintain references to objects in other domains, known as *phantoms*. If three users from Domain B are members of a group in Domain A, the Infrastructure master on Domain A is used to maintain references to the phantom Domain B user members. These phantoms are not manageable or even visible through ordinary means; they are an implementation construct to maintain consistency.

The infrastructure master FSMO role owner is used to continually maintain the phantoms whenever the objects they refer to are changed or moved in the object's domain. When an object in one domain references an object in another domain, it represents that reference by the GUID, the SID (for references to security principals), and the DN of the object being referenced. The Infrastructure master FSMO role holder is the DC responsible for updating an object's SID and distinguished name in a cross-domain object reference.

The infrastructure master is also responsible for fixing up stale references from objects in its domain to objects in other domains ("stale" means references to objects that have been moved or renamed so that the local copy of the remote object's name is out of date). It does this by comparing its (potentially stale) naming data with that of a Global Catalog, which automatically receives regular replication updates for objects in all domains and hence has no stale data. The Infrastructure master writes any updates it finds to its objects and then replicates the updated information around to other DCs in the domain. However, if a GC also holds the Infrastructure master role, by definition the server hosting the GC will always be up to date and will therefore have no stale references. If it never notices that anything needs changing, it will never update any non-GC servers with infrastructure updates.

Once the Active Directory Recycle Bin has been enabled, the infrastructure master's functions are performed independently by every DC in the forest. That is, the tasks just described are no longer delegated to a single DC.

The infrastructure master is additionally responsible for performing updates to the domain when upgrading to Windows Server 2003 or newer—the command *adprep /domainprep* must be run on the infrastructure master. We discuss *adprep* in Chapter 19.

The placement of the infrastructure master and whether or not it can be placed on a Global Catalog without causing issues is often a source of great confusion. Table 2-2 provides a matrix of permitted permutations for forests where the Active Directory Recycle Bin is not enabled.

Table 2-2. Infrastructure master placement rules

	Single-domain forest	**Multiple-domain forest**	
		All domain controllers are GCs	All domain controllers are *not* GCs
Infrastructure master relevant	No	No	Yes
Infrastructure master permitted on GC	Yes	Yes	No

An infrastructure master technically exists for each application partition in the forest, in addition to domains. Prior to Windows Server 2008, the infrastructure master did not perform any functions for application partitions. However, Windows Server 2008 and newer setup versions now enforces a consistency check to make sure that the specified infrastructure master for each application partition is valid. For more information on this, reference microsoft.com. Application partitions are covered in detail in Chapter 4.

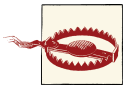
FSMO roles can be transferred between domain controllers. You can transfer the domain naming FSMO with the Active Directory Domains and Trusts snap-in, the schema FSMO with the Active Directory Schema snap-in, and the RID, infrastructure, and PDC emulator FSMOs using the Active Directory Users and Computers snap-in or with Windows PowerShell. Alternatively, you can use the *ntdsutil* utility to perform transfers from a command line. For more information on using *NTDSUTIL* to transfer FSMO roles, see Chapter 18.

Although the AD snap-ins and *NTDSUTIL* can trivially transfer a role from one server to another while both servers are available, there will be some cases in which a FSMO role owner becomes unavailable without the role previously being transferred. In this case, you have to use *NTDSUTIL* to force an ungraceful transfer of the role to a server, known as "seizing" the role. When you seize a FSMO role, you should not bring the original role owner back online. Instead, you should perform a metadata cleanup and then rebuild the domain controller. For more information on using *NTDSUTIL* to seize FSMO roles, see Chapter 18.

If you are using the Windows Server 2008 or newer version of Active Directory Users and Computers, you can delete a domain controller from the Domain Controllers OU and metadata cleanup will be performed.

For earlier versions, in order to remove the metadata from the directory after a failed *dcpromo*, or if a domain controller cannot be brought back online for any reason, you will need to use *NTDSUTIL*. See microsoft.com for more details about the steps involved in using *NTDSUTIL* to perform a metadata cleanup.

If you lose one of the FSMO role holders for a domain, you should always make sure that you are in control of the situation and are promoting a new DC to be the relevant role holder, forcibly moving the role to an existing DC or swiftly bringing back the DC that is the relevant role holder.



If a server with a role becomes unavailable, another server is not automatically promoted to assume the role. The administrator must move the role to a new owner manually.

Many Active Directory administrators spend a great deal of time worrying about the well-being of their FSMO role owners and the "what-if" scenario of scrambling to bring them back into service if one of the role holders goes offline. It is worthwhile to consider just how important the immediate availability of each FSMO role owner is to your environment:

Schema master
: The schema master is only necessary when you are making changes to the schema. These are generally planned well in advance, so if your schema master goes offline you can afford to wait before bringing it back online.

Domain naming master
: The domain naming master is only necessary when adding domains and application partitions. This is another change that is planned well in advance, so again, if your domain naming master goes offline you can probably wait to bring it back online.

Infrastructure master
: If the infrastructure master is offline, you can't run *adprep /domainprep*, and cross-domain phantom updates will stop. The first task is, of course, planned well in advance. The second seems important at first glance, but the infrastructure master updates phantoms over the period of a couple of days, so if you wait a bit to bring it back online, chances are you'll be fine. If you have enabled the Active Directory

Recycle Bin, an infrastructure master outage will only affect your ability to run *adprep*.

RID master
: If the RID master is offline, you can't issue RID pools to DCs when they're requested. Recall from the earlier description of the RID master that domain controllers request RID pools in blocks of 500, when they get down to 250 RIDs remaining. So, unless you expect to exhaust the RID pools on your domain controllers very rapidly (where "rapidly" is faster than you restore the RID master), you're probably not going to have any issues if the RID master is offline for a period of time. One scenario where RID master availability could be more important is if you are provisioning large numbers of new security principals (users, groups, or computers) and the provisioning system targets a single domain controller or set of domain controllers for this task. For more information about RID pool exhaustion, reference this article.

PDC emulator
: The importance of the availability of the PDC emulator varies from environment to environment. The PDC emulator is the domain controller that applications that use legacy APIs will often contact; it is also how trust paths are resolved, how passwords are chained, where the time sync hierarchy is rooted, and so forth. Whether or not you should rush to bring the PDC emulator back online is really quite subjective to your environment, but generally speaking, out of the five FSMO roles, the PDC emulator is probably the most important role holder in your environment.

The fSMORoleOwner Attribute

The FSMO role owners are stored in Active Directory in different locations, depending on the role. The DN of the server holding the role is actually stored as the `fSMORoleOwner` attribute of various objects. For the *mycorp.com* domain, here are the containers that hold that attribute for their respective FSMO roles:

- PDC Emulator - `dc=mycorp,dc=com`
- Infrastructure Master - `cn=Infrastructure,dc=mycorp,dc=com`
- RID Master - `cn=RID Manager$,cn=System,dc=mycorp,dc=com`
- Schema Master - `cn=Schema,cn=Configuration,dc=mycorp,dc=com`
- Domain Naming Master - `cn=Partitions,cn=Configuration,dc=mycorp,dc=com`

The information in the attribute is stored as a DN, representing the NTDS Settings object of the domain controller that is the role owner. So, example contents for this attribute might be:

```
CN=NTDS Settings, CN=MYSERVER1, CN=Servers, CN=My Site,
CN=Sites, CN=Configuration, DC=mycorp, DC=com
```

FSMO role placement has been a subject of some debate over the years. Some administrators advocate placing each role on a different DC, while others advocate keeping all roles together. For the sake of simplicity, you can keep the roles together on a single DC in each domain unless the load on the FSMO role holder DC demands splitting them up onto different servers. If you decide to split them up, see the Microsoft support page for the latest guidance on how to best place these roles.

If you are concerned about being able to restore FSMO role holders from a backup, you should split the roles accordingly. It is specifically a bad idea to restore the RID master from a backup, so you should keep the RID master on a separate domain controller if you want to be able to restore the other FSMO role holders from a backup.

Time Synchronization in Active Directory

Active Directory is highly dependent on all of the domain controllers and domain members having synchronized clocks. Kerberos (which is the underlying authentication protocol for Active Directory clients) uses system clocks to verify the authenticity of Kerberos packets. By default, Active Directory supports a tolerance of plus or minus five minutes for clocks. If the time variance exceeds this setting, clients may be unable to authenticate and, in the case of domain controllers, replication will not occur. Newer versions of Windows can handle time skew without impacting Kerberos, as discussed in this article.

Fortunately, Active Directory and Windows collectively implement a time synchronization system based on the *Network Time* Protocol (NTP) that ensures that every machine in the forest has a synchronized clock. The *w32time* service implements time synchronization on every Windows 2000 or newer machine in the forest. The time synchronization hierarchy is outlined in the following list, and one possible effective hierarchy is shown in Figure 2-5:

1. The forest root domain's PDC emulator synchronizes its clock with a reliable outside time source (such as a hardware clock, a government source, or another reliable NTP server).
2. Each child domain's PDC emulator synchronizes its clock with a reliable time source in its domain or the parent domain.

3. Each domain controller synchronizes its clock with the PDC emulator of its domain or the parent domain.
4. Each domain member synchronizes its clock with the domain controller to which it authenticates.

The Network Time Protocol was defined in RFC 1305, which is available at this link.

You should not need to configure the *w32time* service on any domain joined machine other than your root domain's PDC emulator. While it is permissible to do so, our experience has been that many organizations that elect to use a different time sync hierarchy (such as using local routers or dedicated NTP servers) end up suffering from Kerberos issues later. For information on how to configure the *w32time* service on the PDC emulator, see the upcoming sidebar "Configuring W32Time on the PDC Emulator" on page 23.

Configuring W32Time on the PDC Emulator

In order to configure the PDC emulator, you will need to identify one or more authoritative external time sources. For this example we will use the NTP Pool Project's (*http://www.pool.ntp.org*) NTP servers:

```
w32tm /config /update
  /manualpeerlist:"0.pool.ntp.org,1.pool.ntp.org,2.pool.ntp.org"
  /syncfromflags:manual /reliable:YES
w32tm /resync /rediscover /nowait
```

Some organizations opt to use a local network device such as a router or switch in lieu of an external NTP server. This is a perfectly valid configuration so long as the router or switch is synchronizing with an authoritative source.

For more information on configuring the *w32time* service in this scenario, reference this link. You may also wish to subscribe to the blog located at msdn.com.

For troubleshooting time sync issues, the *w32time* service will log events to the System event log. The *w32tm /monitor* and *w32tm /stripchart /computer:<TargetMachine Name>* commands are often useful for troubleshooting as well.

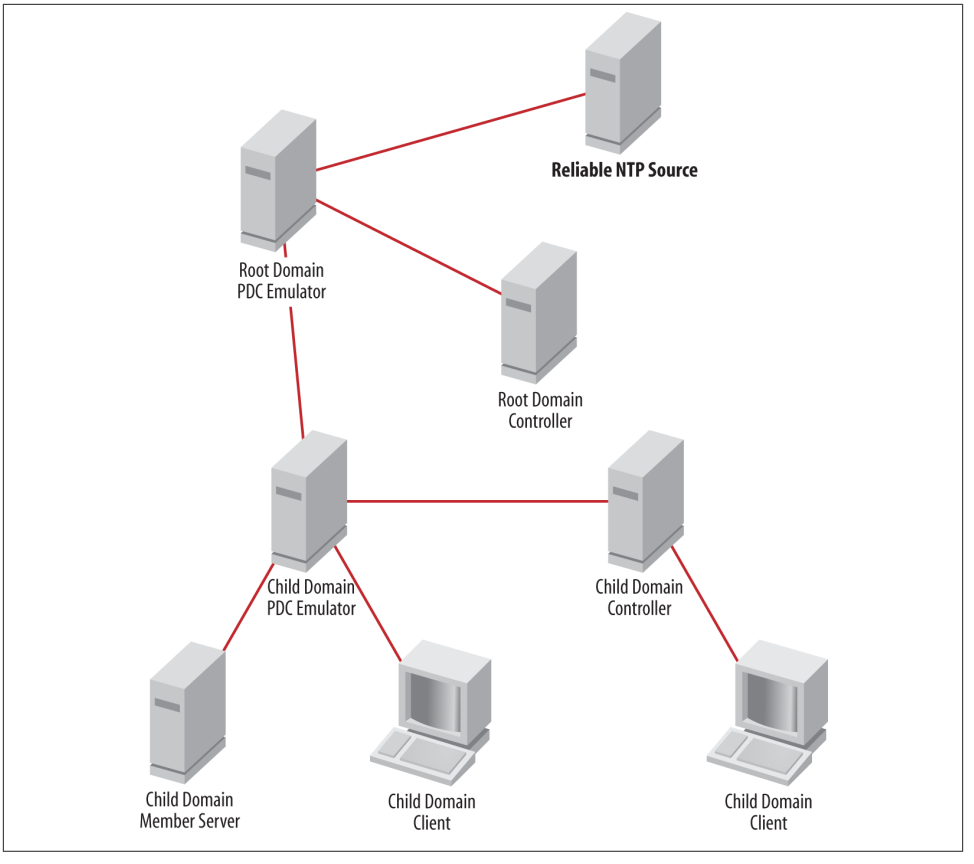
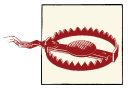


Figure 2-5. Sample time synchronization hierarchy



It is crucial that you remember to reconfigure the *w32time* service on any domain controller that you transfer the PDC emulator FSMO role to or from in your root domain.

We recommend that you consider proactively configuring all of the possible candidates for holding the PDC emulator FSMO role to synchronize with an external reliable time source ahead of time. This will remove the need to remember to perform the manual configuration steps when the role is transferred.

Domain and Forest Functional Levels

For the Windows Server 2003 release of Active Directory, Microsoft expanded on the domain mode concept by introducing *functional levels*. Whereas the domain modes applied only to domains, functional levels apply to both forests and domains. Like the domain mode, functional levels dictate what types of operating systems can assume the

role of a domain controller in a domain or forest. Each functional level also has an associated list of features that become available when the domain or forest reaches that particular functional level. We cover many of the features that are available for each functional level in Appendix A.

Like domain modes, functional levels can be set via the Active Directory Domains and Trusts snap-in. For the Windows Server 2003 and Windows Server 2008 functional levels, once the functional level is "elevated" to a higher status, it cannot be rolled back. Windows Server 2008 R2 introduced the ability to roll back the functional level through the introduction of optional features that are enabled independently of functional levels.

Table 2-3 and Table 2-4 show the operating systems that are supported by the various domain and forest functional levels.

Table 2-3. Domain functional levels

Functional level	Functional level number	Supported domain controller OS
Windows 2000 Mixed	N/A	Windows NT 4.0 Windows 2000 Windows Server 2003
Windows 2000 Native	0	Windows 2000 Windows Server 2003 Windows Server 2008 Windows Server 2008 R2
Windows Server 2003 Interim	1	Windows NT 4.0 Windows Server 2003
Windows Server 2003	2	Windows Server 2003 Windows Server 2008 Windows Server 2008 R2 Windows Server 2012
Windows Server 2008	3	Windows Server 2008 Windows Server 2008 R2 Windows Server 2012
Windows Server 2008 R2	4	Windows Server 2008 R2 Windows Server 2012
Windows Server 2012	5	Windows Server 2012

Table 2-4. Forest functional levels

Functional level	Functional level number	Supported domain controller OS
Windows 2000	0	Windows NT 4.0 Windows 2000 Windows Server 2003 Windows Server 2008 Windows Server 2008 R2

Functional level	Functional level number	Supported domain controller OS
Windows Server 2003 Interim	1	Windows NT 4.0 Windows Server 2003
Windows Server 2003	2	Windows Server 2003 Windows Server 2008 Windows Server 2008 R2 Windows Server 2012
Windows Server 2008	3	Windows Server 2008 Windows Server 2008 R2 Windows Server 2012
Windows Server 2008 R2	4	Windows Server 2008 R2 Windows Server 2012
Windows Server 2012	5	Windows Server 2012

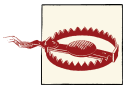
For more information on upgrading Active Directory, check out Chapter 19.

Windows 2000 domain mode

Each Windows 2000 Active Directory domain runs in one of two modes: mixed mode (the default) or native mode. A mixed-mode domain allows servers running previous versions of Windows NT to exist as domain controllers in the domain. A native-mode domain supports only Windows 2000 or later domain controllers. Supporting a mixed-mode domain is necessary to allow administrators to update Windows NT domains to Active Directory. A mixed-mode Active Directory domain emulates some of the behaviors of a Windows NT domain. Remember that with previous versions of Windows NT, networks used to have a primary domain controller (PDC) for a domain that held a writable copy of the accounts database, and zero or more backup domain controllers (BDCs) that held a read-only accounts database copied from the PDC. For an Active Directory network to support older Windows NT domain controllers, one (and only one) of the Active Directory servers has to act as a PDC. The Windows NT PDC notifies the BDCs when a change occurs, and the BDCs then request a copy of the accounts database from the PDC to get the relevant user, group, and computer accounts from Active Directory. While all accounts are replicated out, the total attributes for each object are a much smaller subset of the total attributes that Active Directory now holds for these types of objects.

Going from mixed mode to native mode is a simple, but one-way, change. Since this is a one-way change, you should test it in your lab and plan accordingly. Once you have decided to move forward with the procedure, you simply connect to a DC with the Active Directory Domains and Trusts snap-in and change the mode under the General

tab to native mode. After you have done this, the only way to go back is to reinstall all domain controllers of the domain and restore from a backup made prior to the upgrade.



Never upgrade to Windows 2000 native mode unless you are certain that you will not require any BDCs to exist anywhere in that domain.

The specific differences between mixed mode and native mode are shown in Table 2-5.

Table 2-5. Differences between mixed mode and native mode

Action	Windows 2000 mixed mode	Windows 2000 native mode
Replication	PDC FSMO master sends updates to Windows NT BDCs; same DC acts like ordinary Active Directory DC when communicating with other Active Directory DCs. All Active Directory DCs use multimaster replication between themselves.	Only Active Directory DCs allowed, so all DCs use multimaster replication.
NetBIOS	Can't disable.	Can disable.
Group functions	Windows NT group nesting rules; same-scope group nesting disallowed for global and domain local groups. Domain local groups limited to DCs. Universal groups cannot be security enabled, but can be nested in other universal groups. No conversion between group types or scopes.	Windows AD group nesting rules; same-scope group nesting allowed. Domain local groups available on all domain members. Universal groups can be security enabled. Conversion between group types and scopes allowed.
Account migration	`sIDHistory` is not available; cannot use `Movetree` or ADMT to move objects into the domain.	`sIDHistory` is available. `Movetree` and ADMT can be used.

Moving any domain from mixed mode to native mode has no bearing in any way on any other domain. It doesn't matter if it is the root domain or a subdomain you are converting, because you are only removing the ability of that domain to replicate data to older Windows NT BDCs within the domain, not affecting its ability to replicate and interact with Windows 2000 domain controllers in other domains.

Groups

Active Directory supports three group scopes: *domain local*, *domain global*, and *universal*. Groups in each of these scopes behave slightly differently based on the domain and forest functional levels. To complicate matters further, each group scope can have two types: *distribution* and *security*.

The type is the easiest piece to define. If the type is distribution, the group's SID is not added to a user's security token during logon, so it cannot be used for Windows security purposes. Distribution groups are generally used as a messaging list (a set of users that

you can mail or send instant messages to all at once), though it is possible to use them for security groups for LDAP-based applications or for other applications that don't use the standard Windows security model. Microsoft Exchange represents distribution lists with Active Directory distribution groups. Security groups, by contrast, are enumerated during logon, and the SIDs of any groups of which the user is a member are added to the user's security token. Security groups can also be leveraged by Exchange as distribution lists.

All Windows editions that support Kerberos will encounter problems if security principals are members of too many groups. The issue is that the token of the security principal becomes too large for Windows to handle, and users may experience authentication or other Kerberos issues. This phenomenon is often referred to as *token bloat*. For more information on token size issues, reference this link. The three different scopes of mailing lists and security groups result from the legacy of Windows NT and the introduction of the GC. Global groups and domain local groups are the direct descendants of Windows NT groups; the membership of these groups is only available from domain controllers of the domains in which they are created. Universal group membership is available both from the domain controllers of the domains in which they are created in and from all Global Catalogs in the forest. Universal and global groups can be used in access control lists (ACLs) on any resource in the forest or in trusting domains. Domain local groups can only be used in ACLs in the domain in which they are created.

In order to help you fully understand how groups work in Active Directory, we will explain the following items in this section:

- Which security principals each group type may contain
- How you can nest groups across domain boundaries
- What options are available to you for converting between different group scopes

To start with, let's take a look at what rules govern nesting of groups within a domain. Table 2-6 shows which types of groups can be nested within other types of groups.

Table 2-6. Restrictions on group membership based on group scope

		Can contain domain local		Can contain domain global		Can contain universal	
Scope	**Type**	**Distribution groups**	**Security groups**	**Distribution groups**	**Security groups**	**Distribution groups**	**Security groups**
Domain local	Distribution groups	Yes	Yes	Yes	Yes	Yes	Yes
	Security groups	**Yes**	**Yes**	Yes	Yes	Yes	**Yes**

Scope	Type	Can contain domain local		Can contain domain global		Can contain universal	
		Distribution groups	Security groups	Distribution groups	Security groups	Distribution groups	Security groups
Domain Global	Distribution groups	No	No	Yes	Yes	No	No
	Security groups	No	No	**Yes**	**Yes**	No	No
Universal	Distribution groups	No	No	Yes	Yes	Yes	**Yes**
	Security groups	No	No	**Yes**	**Yes**	**Yes**	**Yes**

Although this table is fine, there is one other complicating factor that needs to be taken into account: cross-domain group membership.

Group membership across domain boundaries

Restrictions for all groups are shown in Tables 2-7 and 2-8. Note that while universal groups can contain members from different domains, those domains cannot span forests. All universal group members must be from the same forest.

Table 2-7. Restrictions on group membership based on group scope

Group scope	Can contain users and computers from		Can contain domain local groups from	
	Same domain	Different domain	Same domain	Different domain
Domain local groups	Yes	Yes	Yes	No
Domain global groups	Yes	No	No	No
Universal groups	Yes	Yes	No	No

Table 2-8. Restrictions on group membership based on domain

Group scope	Can contain domain global groups from		Can contain universal groups from	
	Same domain	Different domain	Same domain	Different domain
Domain local groups	Yes	Yes	Yes	Yes
Domain global groups	Yes	No	No	No
Universal groups	Yes	Yes	Yes	Yes

Tables 2-7 and 2-8 work in conjunction with Table 2-6. You would normally check which groups may be members from Table 2-6 (if any) and then cross-reference with Tables 2-7 and 2-8 to identify what options you have across domain boundaries.

Converting groups

There are limits on what groups can be converted, based on the existing members of the group and the current type and scope of the group. The former should be fairly obvious, based on the existing restrictions shown in Table 2-6. The conversion process cannot work if the existing group members would not be valid members of the new group type once the conversion has taken place. The ability to convert between groups is based on these restrictions:

- Security groups can be converted to distribution groups.
- Distribution groups can be converted to security groups.
- A domain local group can be converted to a universal group provided that the domain local group is not already a member of another domain local group.
- A domain global group can be converted to a universal group provided that the domain global group does not contain any other domain global groups.
- A universal group can be converted to a domain global group provided all members in the group are users from the domain in which the group existed.
- A universal group can be converted to a domain local group.

The benefit of converting a distribution group to a security group is probably obvious: you get to use the group for Windows security purposes. The benefit of converting a security group to a distribution group is usually not so obvious. The most useful aspect of this conversion is that you can safely disable a security group to verify whether or not it is being used for Windows security.

Previously, if you didn't know whether a group was being used for Windows security, you would have to delete it and hope that nothing broke. If anything did break, you found yourself figuring out how to restore the group or how to use a new group. Now you can simply convert the group to a distribution group; and if anything breaks, you simply change the group back into a security group, thereby restoring the old functionality.

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, and forest and domain 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 the management tools available to you for administering Active Directory.

CHAPTER 3

Active Directory Management Tools

Out of the box, Windows is equipped with a multitude of tools for managing Active Directory. Unfortunately, unless you know which tool is suited to your task, it can be challenging to figure out how to get something done without digging through various consoles. Even worse, sometimes there is no GUI for performing a task—leaving you to figure out the appropriate command-line utility or PowerShell command.

This chapter aims to give you a tour of the various management tools that come with Active Directory so that you have a general idea of how to accomplish a task. Throughout this book, you will find references to management tools—both GUI and command-line—and descriptions of how to perform specific tasks with those tools. We won't cover tools that get proper coverage elsewhere in the book in this chapter but will instead direct you to the proper chapter to get the information you need.

Management Tools

The original release of Active Directory in Windows 2000 came with a series of Microsoft Management Console (MMC) snap-ins that enabled GUI-based management of most of the directory. Over time, these snap-ins were improved, but there has not been any major evolution of the Active Directory GUIs that came with Windows 2000. Windows Server 2008 R2 introduced the first version of a new GUI tool for managing Active Directory: the Active Directory Administrative Center (ADAC). ADAC was then further improved upon in Windows Server 2012 to add additional functionality and improve performance.

As we know that not everyone will be running Windows Server 2012 in the near future, this chapter covers both the legacy MMC snap-ins (such as Active Directory Users and Computers) and the new ADAC console. Chances are that you'll find yourself using both toolsets over the long run until ADAC is able to completely replace all of the legacy snap-ins.

Active Directory Administrative Center

The Active Directory Administrative Center is Microsoft's new GUI tool for managing Active Directory. The first version of ADAC came with Windows Server 2008 R2 and the Windows 7 Remote Server Administration Tools (RSAT). There were a number of shortcomings in the initial version of ADAC, and Microsoft corrected many of these with Windows Server 2012 and the Windows 8 RSAT tools.

ADAC is unique when compared to the legacy GUIs in that it is built on top of Windows PowerShell. Any time you do something in ADAC, ADAC calls a PowerShell cmdlet in the Active Directory module for Windows PowerShell. The Windows PowerShell cmdlets in turn connect to the Active Directory Web Service (ADWS) that was introduced in Windows Server 2008 R2. If you want to use ADAC with a domain that does not have any Windows Server 2008 R2 or newer domain controllers, you'll need to download and install ADWS for Windows Server 2003 or Windows Server 2008. We'll discuss ADWS in more detail in Chapter 9, including details on how to add it to older domain controllers.

Figure 3-1 shows a labeled diagram of the ADAC interface, which will acquaint you with the terms used throughout this section (e.g., navigation pane, task pane, etc.). The intuitive right-click menus and contextual options in the task pane will make it easy for you to quickly get up to speed with the interface, so we won't spend much time on the basics of using ADAC for managing objects. Instead, we'll focus on some of the features in ADAC that might not be immediately apparent.

PowerShell History

New to ADAC in Windows Server 2012 is the ability to review the Windows PowerShell commands ADAC executes as you use the console. This is incredibly useful for learning PowerShell, as you can perform a task once graphically and then walk away with the foundation to efficiently script it.

The Windows PowerShell History pane in Figure 3-2 reflects creating a new organizational unit (OU) called North America, and then moving a user object into the North America OU. You can use the plus/minus buttons on the left to expand or collapse a given cmdlet for ease of reading. The Copy button on the toolbar will copy the command to the clipboard so you can reuse it in a script later.

If you want to review all of the read-only PowerShell calls made by ADAC, check Show All (as shown in the upper-right corner of Figure 3-2). This will show you commands executed for searching, for example. This is disabled by default, as the amount of data that will be presented is substantial.

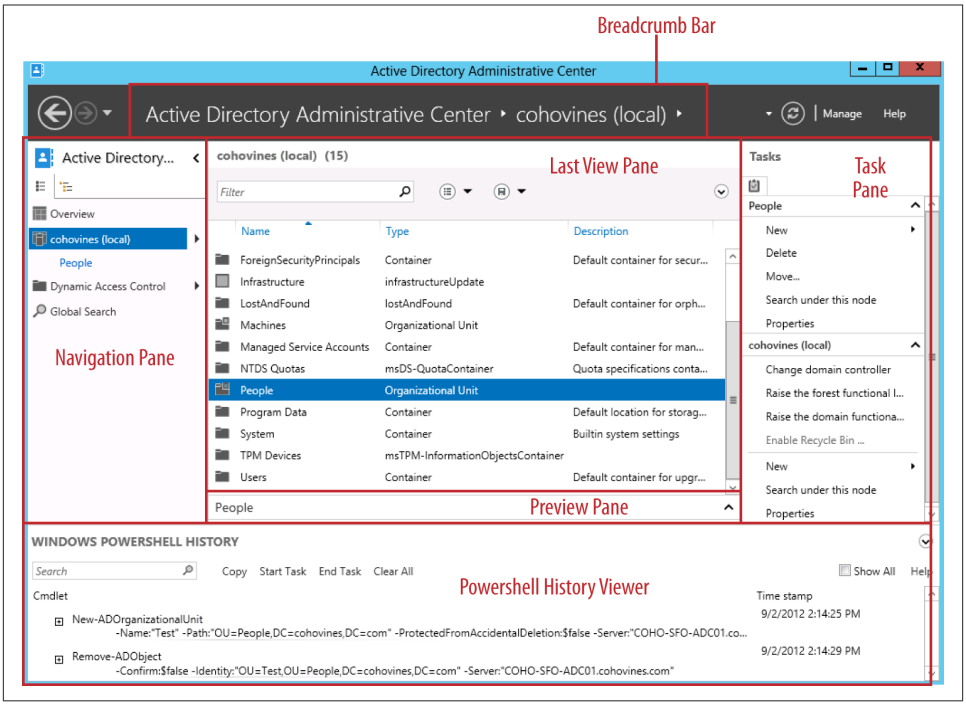


Figure 3-1. Active Directory Administrative Center diagram

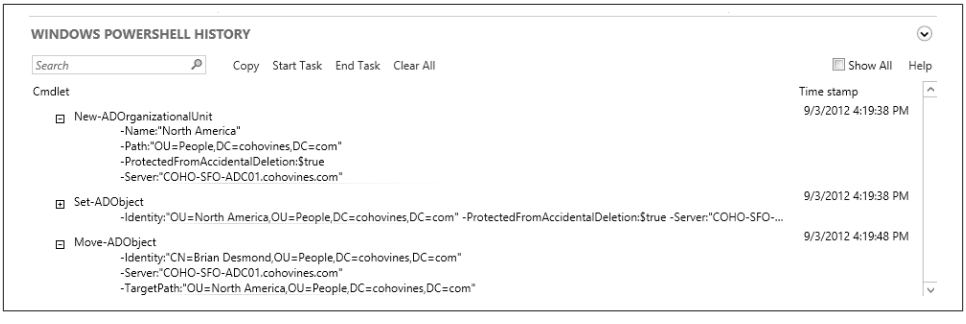


Figure 3-2. The PowerShell History pane

Global Search

The search functionality in ADAC provides a handy way for you to search Active Directory and learn the LDAP search syntax while you're at it. Accessible from the navigation pane under the Global Search node, you can either build a query graphically or enter an LDAP filter directly.

In order to add criteria graphically, you'll need to expand the search area by clicking the down arrow to the far right of the search text box, as shown in Figure 3-3. The "Add criteria" button has a number of common queries that you can choose from, as well as common AD attributes if you scroll further down the list.

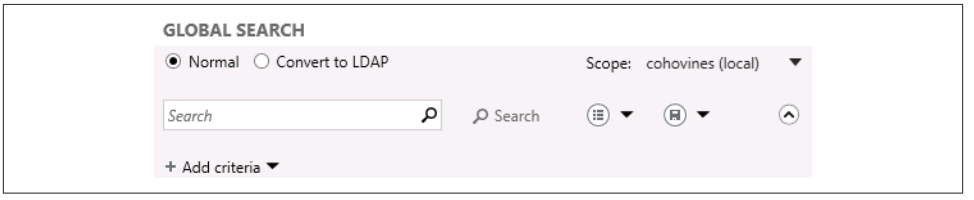


Figure 3-3. Global Search filter builder

Once you have constructed your search, you can view the traditional LDAP syntax making up the query by clicking the "Convert to LDAP" radio button (seen in the upper-left corner of Figure 3-3). In Figure 3-4, ADAC is searching for the list of user accounts that are enabled but also locked out. You can copy the filter to the clipboard and either modify it or use it in another LDAP search program.

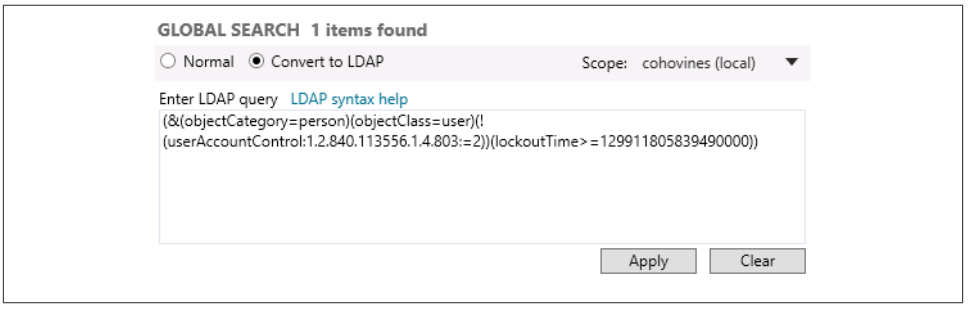


Figure 3-4. Global Search LDAP filter view

Also of interest in the Global Search view, and specifically in Figure 3-3, are a couple of additional options. The first option is the Scope drop-down in the upper-righthand corner. With the Scope drop-down, you can select the domain to search, or target the Global Catalog to search the entire forest. Using the button with the icon of a floppy disk, you can save queries. The queries will be available for later use in ADAC so you don't need to reconstruct them.

If you enter a custom LDAP filter in the "Convert to LDAP" view, you cannot save this query for later use. Only queries built with the criteria builder can be saved for later use.

Multiple-domain support

One of the advantages of ADAC over Active Directory Users and Computers is that it supports simultaneously working with multiple domains in the forest. It may not be immediately apparent how to add domains to ADAC. To add a domain, right-click in the whitespace of the navigation pane, and then click Add Navigation Nodes.

In the lower-righthand corner of the Add Navigation Nodes screen, as shown in Figure 3-5, is a link to "Connect to other domains." Click this link and enter the fully qualified domain name (FQDN) of the domain you want to manage.

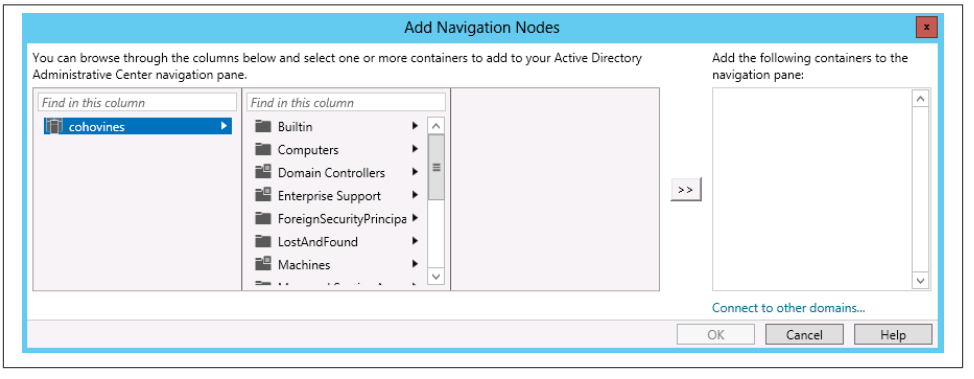


Figure 3-5. Add Navigation Nodes dialog

You can also use the Add Navigation Nodes screen to pin organizational units (OUs) to the navigation pane.

Extensibility

If you've worked with Active Directory Users and Computers (ADUC) before, you may be aware that ADUC can be extended to include additional columns and tabs. Unfortunately, ADAC doesn't expose corresponding functionality. Any additional tabs registered for use with ADUC are somewhat awkwardly shown in the Extensions section of an object's property pages, as shown in Figure 3-6. Of particular interest here are the Security and Attribute Editor tabs, which aren't replicated elsewhere in ADAC.

Active Directory Users and Computers

ADUC has been around since Windows 2000. Figure 3-7 shows the Windows Server 2012 version. If you're a veteran Active Directory administrator, you've probably noticed that not much has changed in ADUC over the course of more than a decade. That said, ADUC remains the go-to tool for most administrators, despite the introduction of

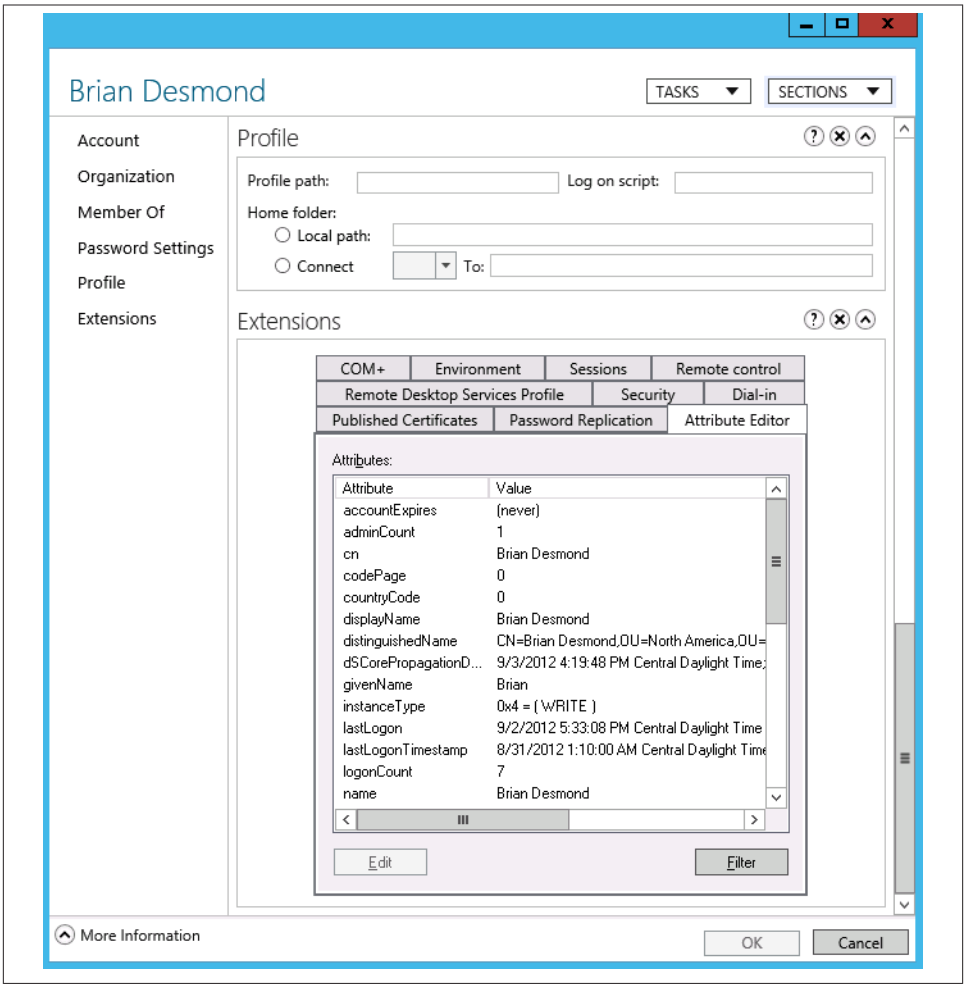


Figure 3-6. Extensions section of an object in ADAC

ADAC. You can launch ADUC from the Administrative Tools section of the Control Panel or Start menu, or, for speed, just run *dsa.msc*.

As a directory management tool, ADUC excels at providing a frontend for managing users and groups, as well as searching and browsing the directory and managing the hierarchy (organizational unit structure). Although ADUC has some other useful features, it generally isn't the best tool for the job for tasks beyond this scope. We won't spend too much time explaining how to go about creating or editing an object in this section, but we'll point out some tips and tricks to help you take advantage of all of the functionality in ADUC.

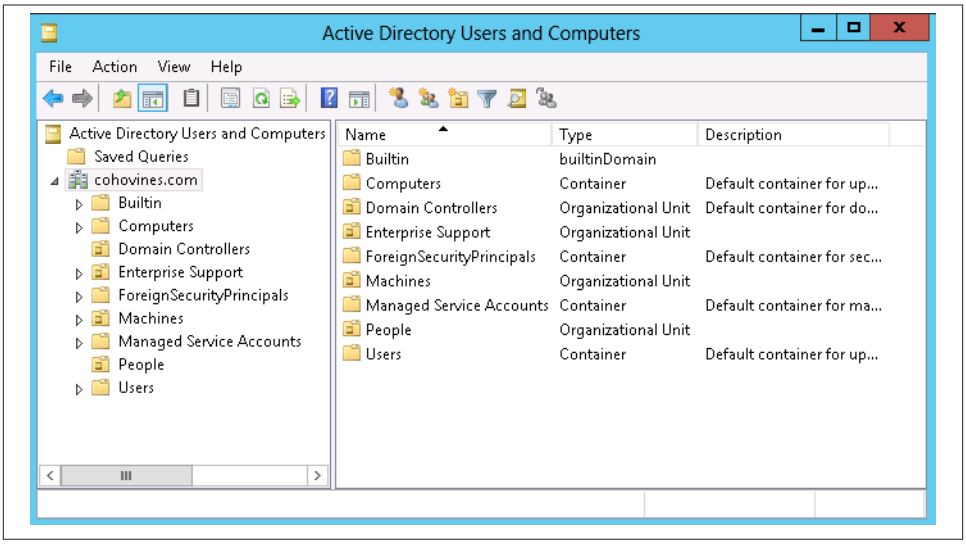


Figure 3-7. Active Directory Users and Computers (Windows Server 2012 version)

In general, the best place to find the features available in any given part of the directory with ADUC is via the right-click menu. ADUC will dynamically filter the options for object creation and management based on where in the directory you are, as well as the permissions you have in that location. ADUC is also smart enough to make text boxes and other controls read-only when you are viewing the properties of an object if your permissions don't allow you to edit a given attribute. For example, in Figure 3-8, the administrator does not have enough access to modify the email attribute of the user account, so ADUC has marked the field as read-only.

Advanced Features

By default, ADUC doesn't show all of the functionality it has to offer. In order to see all of the functionality, you need to make sure that View→Advanced Features is checked. Once you enable this option, you'll see a number of additional containers in the domain as well as numerous additional tabs in the properties of objects. If you compare Figure 3-8 and Figure 3-9, you'll see how many additional tabs are visible on a user account with Advanced Features enabled.

Of particular interest are the Security and Attribute Editor tabs. The Security tab, shown in Figure 3-10, allows you to edit the permissions on a given object using a friendly interface (similar to managing the permissions on the filesystem, for example). We'll discuss managing permissions in Active Directory in depth in Chapter 16. The Attribute Editor tab lets you see all of the attributes of an object, regardless of whether or not

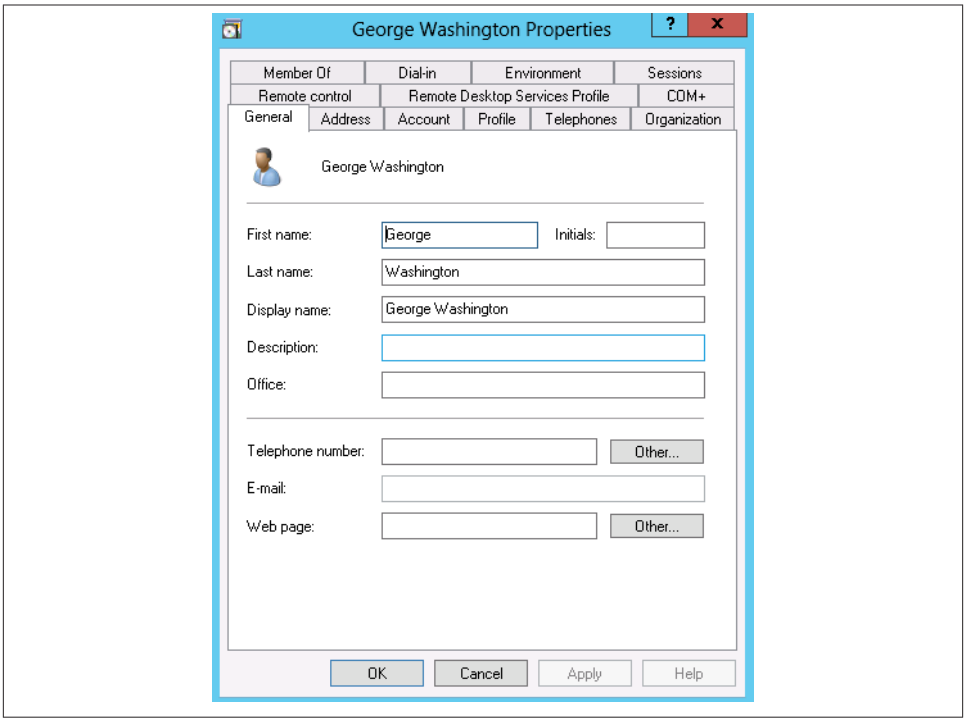


Figure 3-8. Dynamically disabling fields based on permissions

ADUC natively exposes them in one of the property pages. We'll look into the Attribute Editor in more detail in the "ADSI Edit" on page 45 section of this chapter.

Saved Queries

If you work with specific sets of objects across the directory, or you are often searching for the same thing, you may find the Saved Queries feature to be handy. With Saved Queries, you can create custom "views" of the directory that ADUC remembers across sessions. To create a query, right-click the Saved Queries node (as seen in the upper-left corner of Figure 3-7) and click New→Query.

In the New Query window (Figure 3-11), you can provide a name, a query root, and the actual LDAP filter that will find the objects you're looking for. The query root simply allows you to define where in the directory your search should begin. By default, the query will search the entire domain, but you can use the Browse button to scope your query to a specific OU, for example.

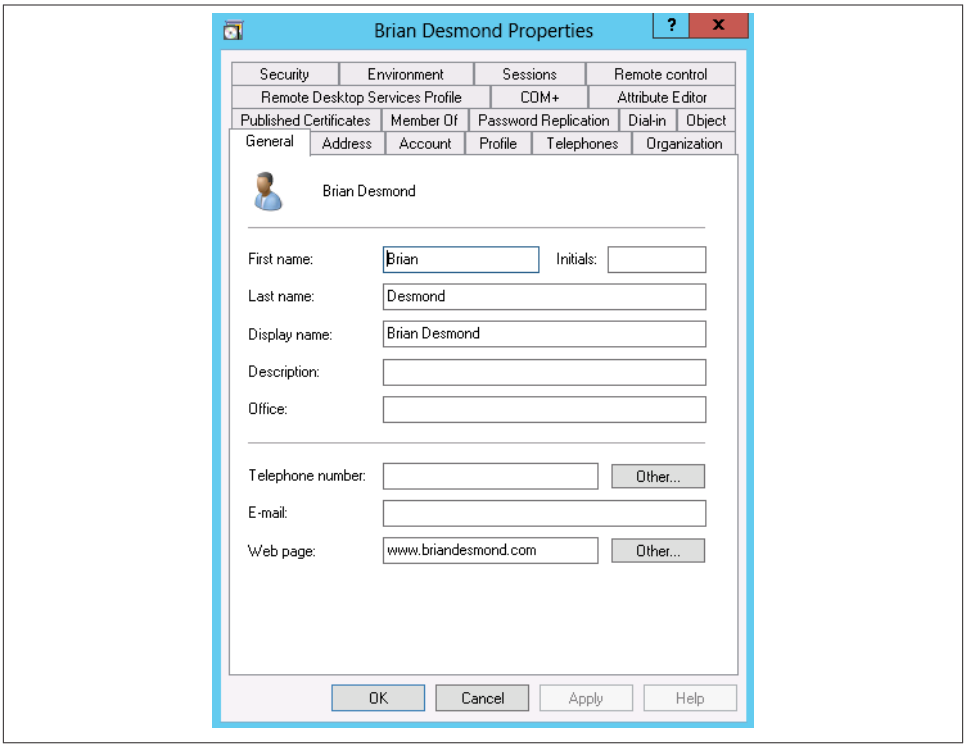


Figure 3-9. User properties with Advanced Features enabled

To construct your search, click Define Query. As you can see in Figure 3-12, ADUC provides a convenient filter builder to graphically define your search so that you won't need to be comfortable with building LDAP filters by hand. If one of the canned templates doesn't work for you, you can switch to the object type you're looking for from the Find drop-down at the top of the screen, or you can use the Custom Search option in the drop-down to enter an LDAP filter directly. We'll discuss building LDAP filters in depth in Chapter 7.

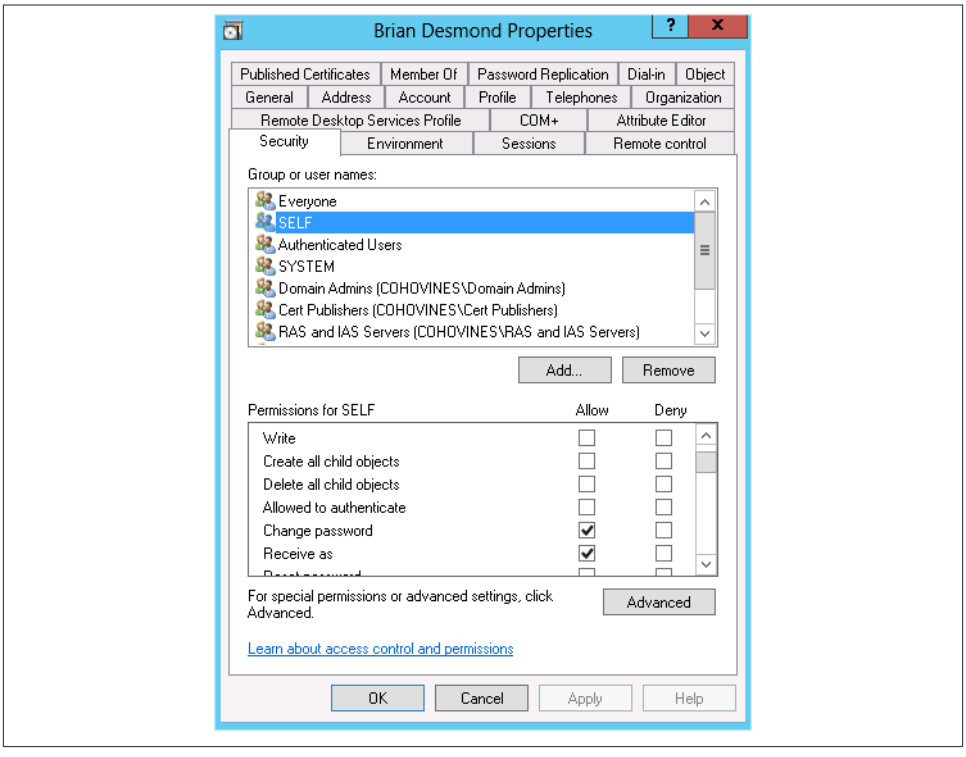


Figure 3-10. The ACL editor on the Security tab

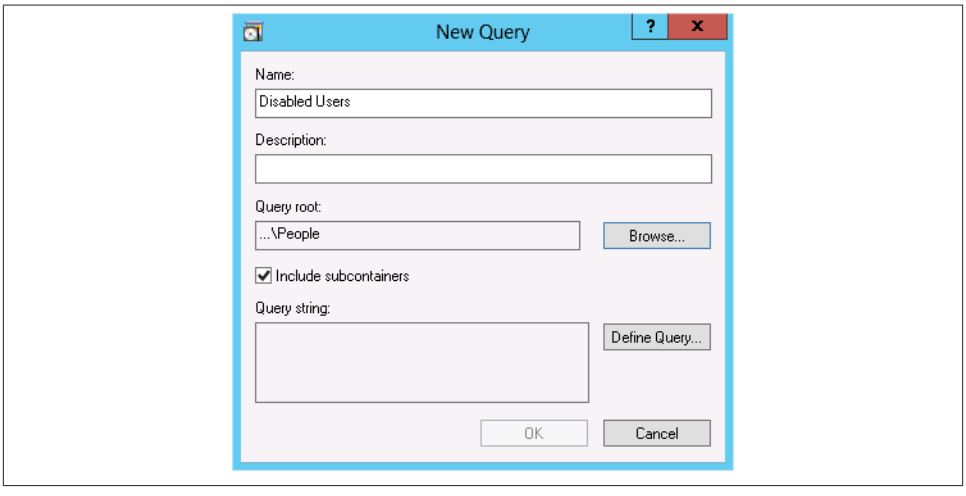


Figure 3-11. New Query dialog

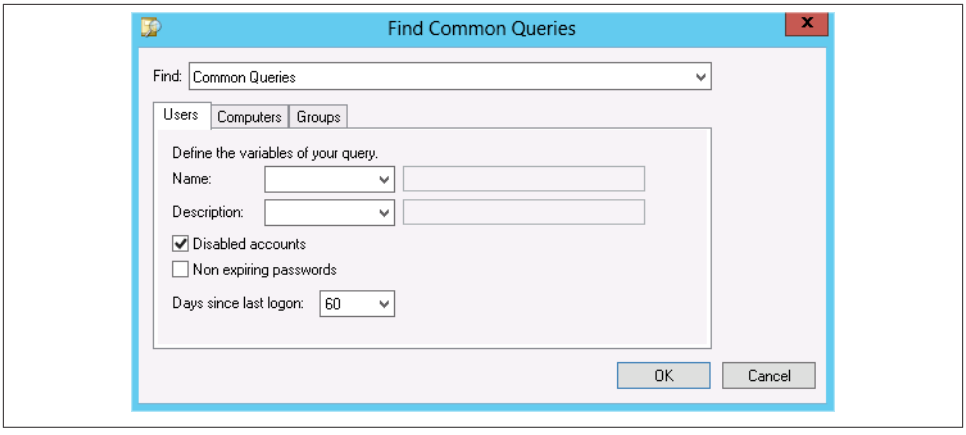


Figure 3-12. LDAP filter builder

The definitions for Saved Queries are stored locally, so if you open ADUC on another machine, you won't see your queries. Fortunately, you can right-click any Saved Query and use the Export Query Definition option to save the query to an XML file. You can then import the XML file on another machine or share it with a colleague.

Controlling drag-and-drop moves

Another handy feature in ADUC is the ability to drag and drop an object or an entire object hierarchy to a new location in the directory. Under the covers, ADUC takes care of the necessary LDAP commands to actually move the object or tree. Unfortunately, the downside of this feature is that it's incredibly easy for an administrator to accidentally move objects. To combat this problem, Microsoft made a couple of changes in later versions of ADUC.

The first change is a warning message that asks the administrator to confirm the move and advises of some of the potential pitfalls of moving objects (Figure 3-13). Unfortunately, reading dialog boxes before clicking OK is not something that many administrators are especially diligent about. To solve this, a setting in Active Directory can be set to disable drag-and-drop moves in ADUC altogether.

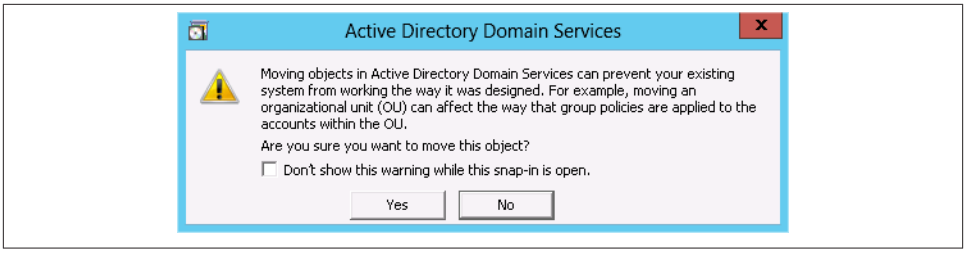


Figure 3-13. Drag-and-drop move object confirmation

To learn to disable drag-and-drop moves, take a look at Microsoft Knowledge Base article 827687. You'll need to use ADSI Edit to make the change; if you're not familiar with this tool yet, take a look at our discussion of ADSI Edit in the next section.

Taskpads

One of the features that ADUC inherits from its roots as an MMC snap-in is the ability to create Taskpads. Taskpads let you create custom views of ADUC with custom subsets of tasks. This can be a great way to create an ADUC console for helpdesk technicians or delegated OU administrators.

To create a Taskpad, you'll need to launch a new MMC console (Start→Run→**`mmc.exe`**), and then add the ADUC snap-in. To add the snap-in, go to File→Add/Remove Snap-in. Find Active Directory Users and Computers on the left side and click Add >. Once you have a new MMC setup, right-click where you want to start the Taskpad and click New Taskpad View, as shown in Figure 3-14.

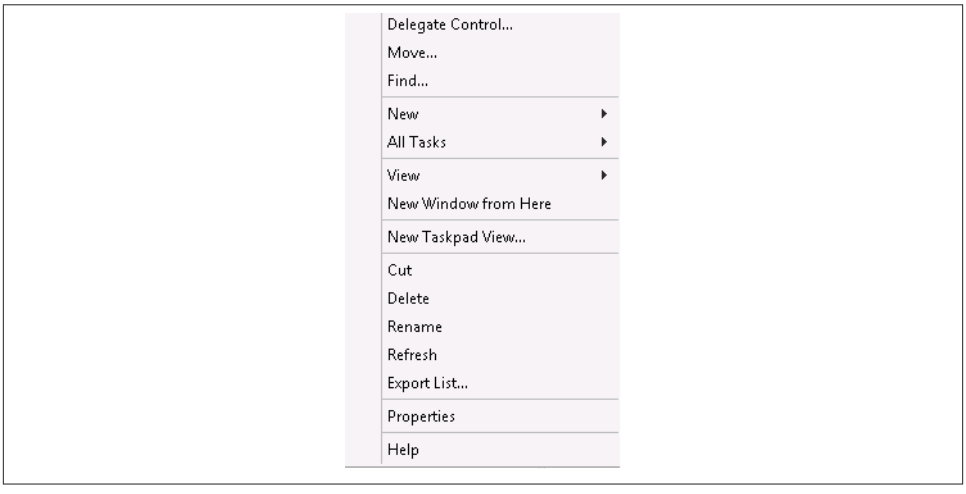


Figure 3-14. Context menu in ADUC

Once you're done customizing with the wizard, be sure to save the console via the File→Save menu. If you're looking for a simple console that starts at a given OU and is otherwise identical to ADUC, use the "New Window from Here" option shown in Figure 3-14. Save the resultant MMC, and each time it's opened ADUC will launch from the location you specified. This latter option is especially useful for a quick solution to providing filtered consoles to junior administrators and technicians.

ADSI Edit

ADSI Edit is a management tool that dates to the early Resource Kit tools for Active Directory. Like many of the Resource Kit tools, ADSI Edit is now included with the RSAT tools for Active Directory; it is accessible via the Administrative Tools section of the Control Panel or by launching *adsiedit.msc*.

The main benefit of ADSI Edit is that it allows you to browse all of the attributes of an object in a friendly (and sometimes decoded) way and to find objects by browsing to them much like you would with ADUC or ADAC. ADSI Edit also lets you view and edit data in the Configuration naming context with a simple GUI interface.

If you're launching ADSI Edit for the first time, the first thing you'll need to do is go to Action→Connect To. From the Connection Settings dialog shown in Figure 3-15, you can pick a naming context from the "Select a well known Naming Context" drop-down, or if you're connecting to an AD LDS instance, enter the distinguished name (DN) of the naming context instead. Likewise, you can specify a specific domain controller or AD LDS server, or you can opt to let ADSI Edit pick one for you. The Advanced button lets you connect to a Global Catalog or specify alternate credentials for binding with.

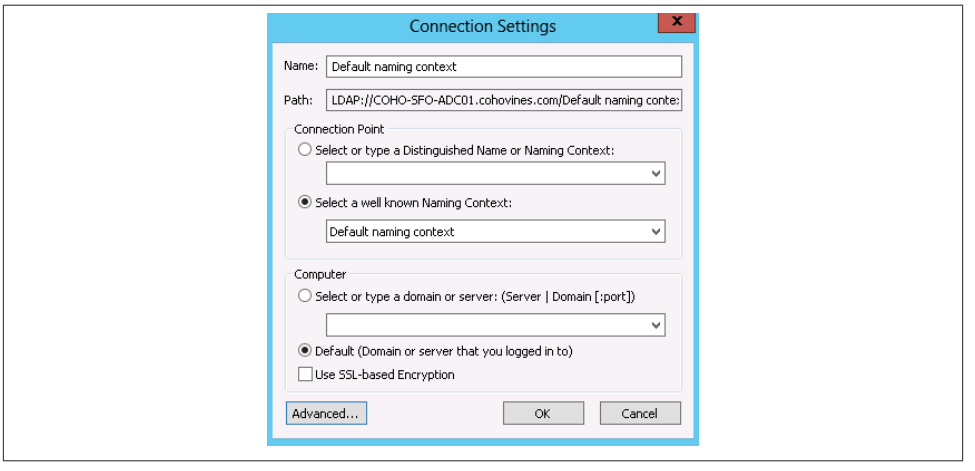


Figure 3-15. ADSI Edit Connection Settings

Once you have connected, you can browse the structure much as you would with any other tool. To review the attributes of an object, right-click on it and then click Properties. You'll get a dialog similar to Figure 3-16.

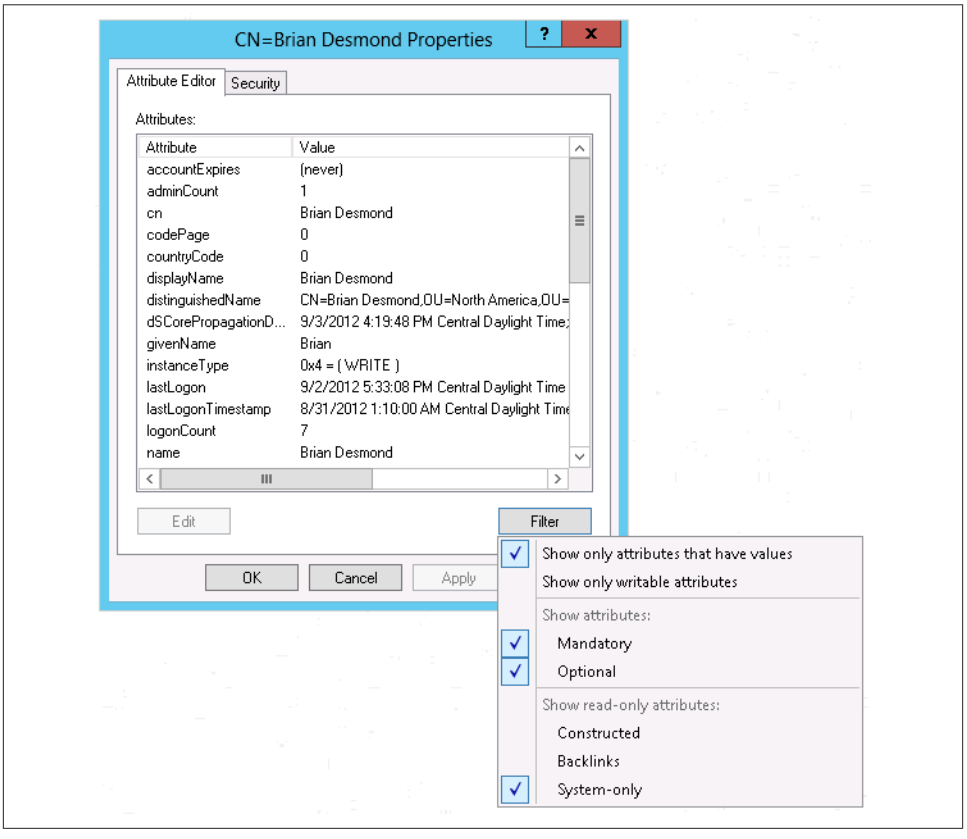


Figure 3-16. ADSI Edit Attribute Editor with Filter menu

Technically you should be able to connect to and manage a third-party LDAP server with ADSI Edit, but in our experience this typically does not work well.

Take note of the Filter button's menu, which is also shown in Figure 3-16. You can use this menu to limit the attributes that are shown and make the output from ADSI Edit more useful. Here are the options in the Filter menu:

Show only attributes that have values
: Filters the list to only show attributes that have been populated with data.

Show only writable attributes
: Filters the list to only show attributes that Active Directory reports are writable by the current user.

Show attributes: Mandatory
: Shows attributes that are marked as mandatory (must be populated) for the object class (e.g., user) in the schema.

Show attributes: Optional
: Shows attributes that are marked as optional (do not need to be populated) for the object class (e.g., user) in the schema.

Show read-only attributes: Constructed
: Displays attributes that have a value computed internally by Active Directory on the fly.

Show read-only attributes: Backlinks
: Displays the backlink component of a linked attribute relationship. Backlink attributes are constructed and not changeable. For example, in a group membership, the `member` attribute on the group is changeable, while the backlink attribute—the `memberOf` attribute on the user—is not.

You can also edit the ACL on an object by clicking the Security tab. The ACL editor in ADSI Edit is the same interface as in ADUC and ADAC. If you right-click on a user, there is a contextual interface to reset the user's password, which is particularly useful when managing an AD LDS instance. You can also right-click and delete objects, move objects (there is no drag-and-drop in ADSI Edit), or create an object.

Creating an object with ADSI Edit may be easier than using other raw LDAP editing tools, but it is not especially straightforward. When you click Create, ADSI Edit will first present you with a list of all the object classes in the schema that can be created in the location you selected. Next, ADSI Edit will present you with a page to specify the value for each mandatory attribute defined on the selected object class in the schema. Finally, you will have the option of specifying optional attributes with the More Attributes button on the last page.

ADSI Edit also includes an LDAP query function accessible by right-clicking and going to New→Query. You can either enter a raw LDAP query or use the Windows LDAP query builder to graphically construct your search.

LDP

Sometimes the easiest way to dig into a problem is to look at the supporting data in a raw format, without any of the abstractions that GUI tools typically provide. Microsoft

has included a tool called LDP with Active Directory since Windows 2000. Originally LDP was included in the Windows Resource Kit Tools, and then it eventually morphed into a tool that's installed when you install the Active Directory RSAT tools. You can use LDP to connect to any LDAP server (even non-Microsoft platforms), perform searches, view data, and make modifications.

You won't find LDP on the Start menu, but you can launch it from a command prompt or the Run dialog by simply typing **`ldp`**. Once LDP starts, you will need to connect to a domain controller (or an AD Lightweight Directory Services instance or another LDAP server). You can do this by clicking Connection→Connect to open the Connect dialog, shown in Figure 3-17. If you want to encrypt your session with SSL, check the SSL checkbox and change the port to 636.

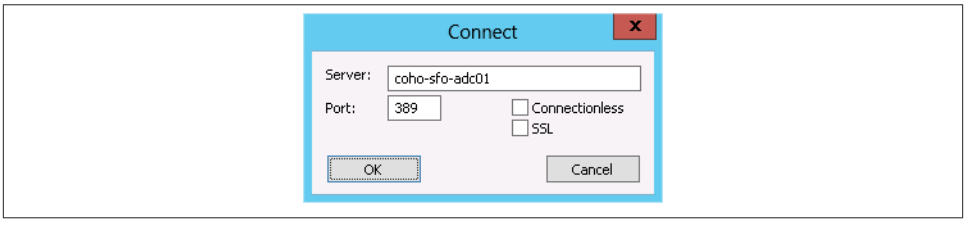


Figure 3-17. LDP Connect dialog

Once you've connected, you'll be presented with some anonymously accessible data about the directory known as the *RootDSE*. If you want to view any of the data stored in the directory, you'll likely need to authenticate your connection. To do this, click Connection→Bind to open the Bind dialog, shown in Figure 3-18.

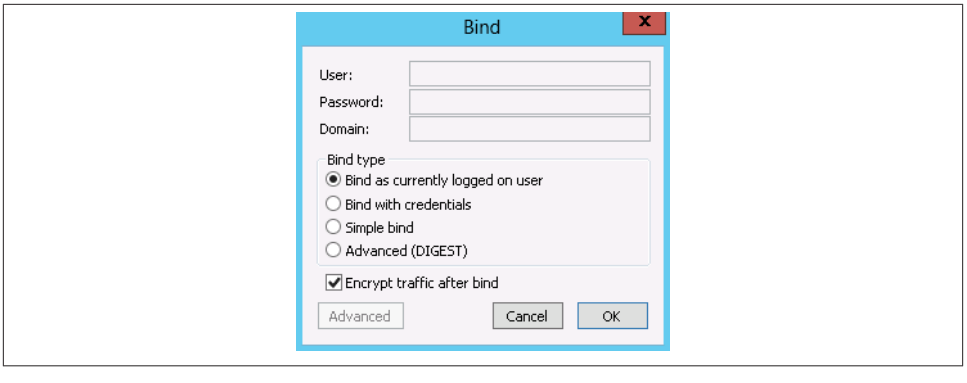


Figure 3-18. LDP Bind dialog

There are a number of options you can choose from in the Bind dialog. Specifically, the bind types are:

Bind as currently logged on user
: This passes your current credentials through. If you are signed in with a domain account with sufficient permissions on the target server/directory, you can generally choose this default option.

Bind with credentials
: This allows you to specify a set of domain credentials to bind to the server as. This option is useful if you are signed in with an account that does not have enough permissions and you need to elevate your connection.

Simple bind
: Simple binds are what many applications use, and this is the choice you will likely make if you are connecting to a third-party LDAP server. Keep in mind that simple binds have no mechanism to protect your credentials—unless you are using LDAP over SSL, the credentials you specify in a simple bind will be passed in clear text over the network.

For the purposes of this discussion, we recommend that you use the default option, "Bind as currently logged on user." Once you bind to the directory, LDP will display the results of the bind attempt in the information pane at the bottom right. A successful bind attempt might generate output similar to the following:

```
0 = ldap_set_option(ld, LDAP_OPT_ENCRYPT, 1)
res = ldap_bind_s(ld, NULL, &NtAuthIdentity, NEGOTIATE (1158)); // v.3
    {NtAuthIdentity: User='NULL'; Pwd=<unavailable>; domain = 'NULL'}
Authenticated as: 'COHOVINES\briandesmond'.
-----------
```

An unsuccessful bind attempt may generate a number of different types of errors, depending on the failure. In the case of bad credentials, you might get an error similar to this:

```
res = ldap_simple_bind_s(ld, 'BadUsername', <unavailable>); // v.3
Error <49>: ldap_simple_bind_s() failed: Invalid Credentials
Server error: 80090308: LdapErr: DSID-0C0903C5, comment: AcceptSecurityContext
  error, data 52e, v23f0
Error 0x80090308 The token supplied to the function is invalid
-----------
```

Once you've bound to the directory, you're ready to start taking advantage of LDP's features. If you simply want to browse the directory and find the object you're looking for, go to View→Tree and enter or select the root from which you want to browse LDP will build a tree on the left side of the screen that you can browse through.

To view the attributes of an object, double-click the object. Figure 3-19 shows the output after double-clicking a user object. You can see that LDP shows all of the attributes of the object in a raw format. LDP will also provide useful hints next to some attributes. For example, the `userAccountControl` attribute is a bit mask that controls multiple settings for a given object. In Figure 3-19, LDP decodes the value (`0x10200`) and tells

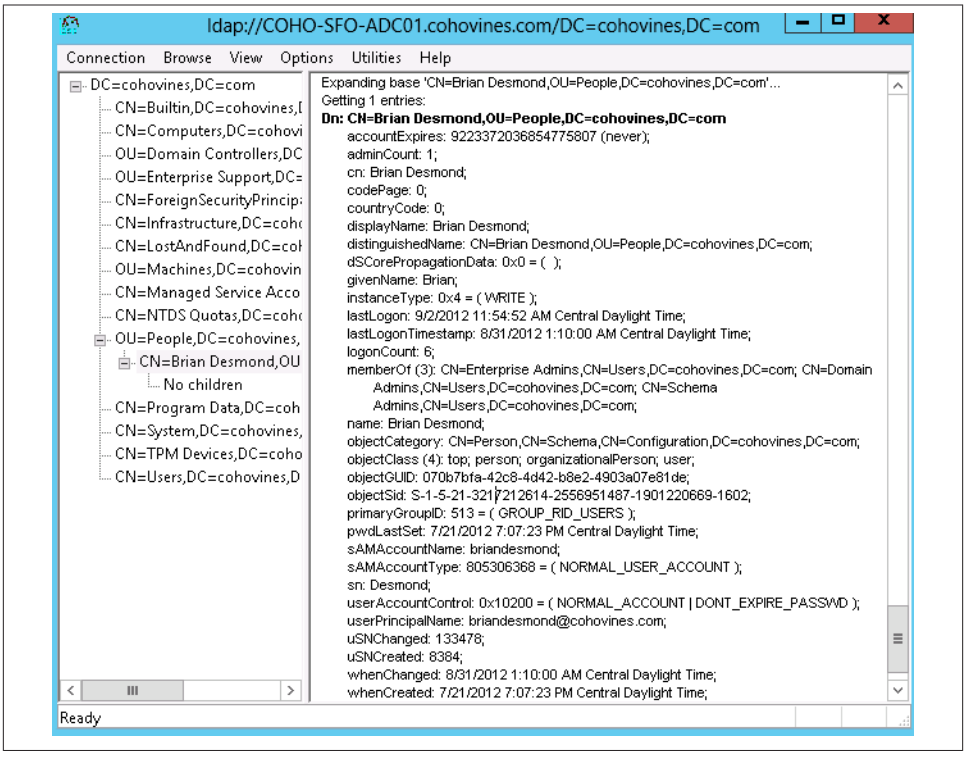


Figure 3-19. LDP main view

you that the object is a normal (enabled) account that is configured to never have its password expire.

If you have a large directory or you know exactly what you're looking for, you can issue an LDAP query to the directory by clicking Browse→Search. Figure 3-20 shows a search being constructed for a computer called *COHO-CHI-ADC01*. You'll need to specify the attributes you want to have returned with the search, or you can enter an asterisk (*) to return all attributes. You can also control numerous advanced options for the search, such as sorting, timeouts, and the use of special LDAP controls via the Options button. We'll discuss many of these options in Chapter 7 when we discuss Active Directory searches in depth.

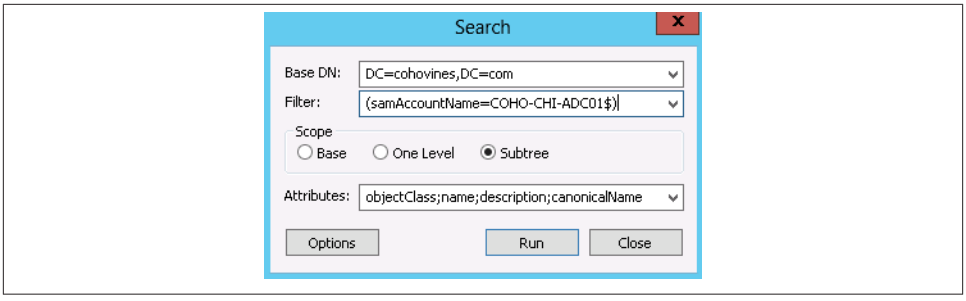


Figure 3-20. LDP Search dialog

To make changes to an object, right-click the object in the tree on the left of LDP and click Modify. You can modify one or more attributes at once by entering the attribute name and value and then clicking the Enter button, as shown in Figure 3-21. In Figure 3-21, we change the `title` attribute on a user account to `Author`.

The choice of operation type is important and can be a little confusing. If you are updating a normal attribute, use the Replace operation. If you want to append a value to a multivalued attribute, use the Add operation. Finally, if you want to clear an attribute or remove a specific value from a multivalued attribute, use the Delete operation. Once you're ready to submit your change, click Run.

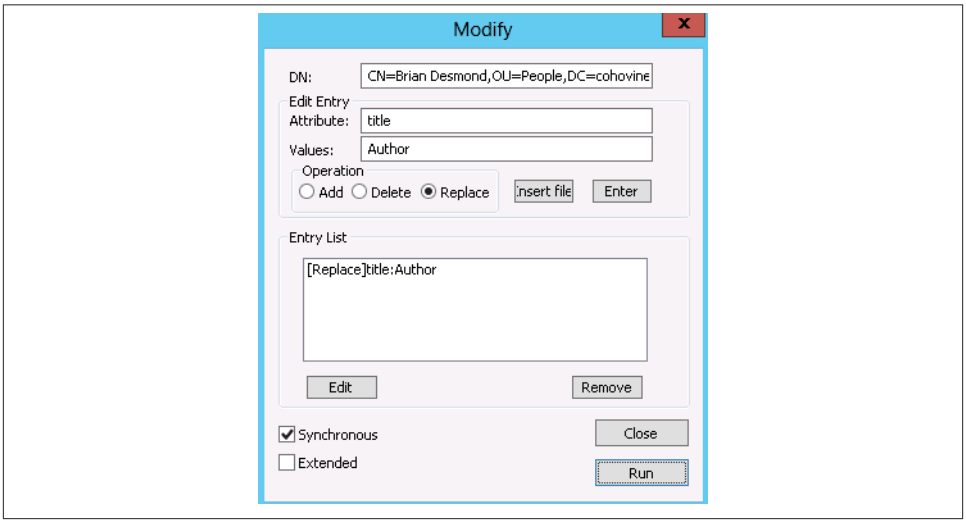


Figure 3-21. LDP Modify dialog

If the modification succeeds, LDP will report success with output similar to the following:

```
***Call Modify...
ldap_modify_s(ld, 'CN=Brian Desmond,OU=People,DC=cohovines,DC=com',[1] attrs);
Modified "CN=Brian Desmond,OU=People,DC=cohovines,DC=com".
-----------
```

If the modification fails, LDP will provide an error code and you can resubmit your request.

There are a number of other useful features in the LDP tool, including the ability to delete an object or tree (Browse→Delete), rename an object (Browse→Modify DN), create objects (Browse→Add Child), view replication metadata (Browse→Replication→Replication Metadata Descriptor, or right-click the object and click Advanced→Replication Metadata), and view the ACL (security descriptor) of an object (Browse→Security→Security Descriptor, or right-click the object and click Advanced→Security Descriptor). We'll cover the security descriptor functionality in LDP in more depth in Chapter 21.

LDP is a powerful tool that you can use to perform nearly any task in an LDAP directory. Taking advantage of LDP's power requires a sometimes deep understanding of the LDAP protocol, but when you need to look at raw data or perform an operation directly on the directory without any abstraction, LDP is an excellent tool to have in your tool belt.

Customizing the Active Directory Administrative Snap-ins

Unfortunately, the Active Directory Administrative Center is not customizable in terms of the data that's available in the console. The legacy snap-ins such as Active Directory Users and Computers are highly extensible, however. One of the most common examples of extending ADUC is to display additional attributes, perhaps even ones you've created. Let's say that you decide to add an attribute called `myCorp-LanguagesSpoken` to the Active Directory schema. In turn, you want `myCorp-LanguagesSpoken` to be displayed in ADUC so others can view the languages a user speaks. Fortunately, the legacy Active Directory snap-ins (e.g., ADUC and AD Sites and Services) are customizable by modifying one or more attributes in Active Directory. You can also extend the functionality of a snap-in using Windows Scripting Host (WSH), Visual Basic (VB), or any other COM-based language.

This section is devoted to reviewing the components behind the Active Directory administrative snap-ins and how you can modify them to meet your needs. These components include:

Display specifiers
: Objects in Active Directory that contain localized user interface information

Property pages
: Tabbed dialog boxes that display information

Context menus
: Menus displayed after right-clicking an object (e.g., a user)

Icons
: Images displayed when viewing a particular class

Display names
: User-friendly names displayed for attributes and classes (e.g., Last Name)

Creation wizard
: A wizard interface used to create an object

Display Specifiers

Display specifiers are objects stored in Active Directory that contain information on how to display and manage objects for a specific object class through the Active Directory snap-ins. These display specifiers are held in the Configuration naming context under the DisplaySpecifiers container. Within the DisplaySpecifiers container, there is a container for each supported locale, in a path similar to this:

```
LDAP://cn=409,cn=DisplaySpecifiers,cn=Configuration,dc=mycorp,dc=com
```

The preceding container contains the display specifiers for the US/English locale of 409. If you wanted to create or manage display specifiers for a different locale, you would just create a new container with the relevant hexadecimal code for the locale and populate it with the relevant display specifier objects. For example, 409 in hex represents 1,033 in decimal, and 1,033 is the US/English locale. If we created 809 (2,057 in decimal), we would get the UK/English locale, and if we created 40C (1,036 in decimal), we would get the French locale. The currently installed locale values can be found in the registry at `HKLM\SYSTEM\CurrentControlSet\Control\ContentIndex\Language`. Having display specifiers per locale enables you to support a wide variety of languages for a geographically dispersed client base.

Each of the locale-specific containers has a series of objects of the `displaySpecifier` class. The object names are in the form of `ObjectClass-Display`. The `user` class has one called `User-Display`, the `computer` class has one called `Computer-Display`, and so on. To extend the interface for a specific object class for a particular language, you just need to modify the appropriate attributes on the `displaySpecifier` object that represents the class in that container.

Here's a simple example. The `classDisplayName` attribute exists on all `displaySpecifier` objects. Let's say we use the ADSI Edit tool from the Support Tools to open up the `Group-Display` object and change this attribute from `Group` to `Moose`. If we right-click on any container in the ADUC tool, a context menu normally appears, allowing us to create a new user, group, or organizational unit (among other things). After making the edit and reopening ADUC, it allows us to create a new user, *moose*, or organizational

unit. The way that the group class was displayed in the interface has been changed. If we wanted to change the display specifier for the French locale as well as or instead of the US/English locale, we would go to (or create) the 40C container and apply the change to the Group-Display object.

Let's now review some of the other customizations you can make.

Property Pages

You can see the array of property pages that exist by opening the properties of any object in ADUC. You can add property pages to these and display your own here. For this to work, though, the property page has to exist as a Component Object Model (COM) object that supports the IShellExitInit and IShellPropSheetExt interfaces. This means that the property page has to be created first in Visual Basic, Visual C++, or something similar.

Creating the object is the hardest part. Actually telling the system to use it is easy. Once the property page COM object exists, it will have a globally unique identifier (GUID). You then use ADSI Edit to go to the display specifier object representing the class that you wish to modify and alter the adminPropertyPages or shellPropertyPages attributes. These attributes are multivalued and store data in the following form:

```
2, {AB4909C2-6BEA-11D2-B1B6-00C04F9914BD}
1, {AB123CDE-ABCD-1124-ABAB-00CC4DD11223}
```

The first item represents the order number in which the sheets should appear. The second represents the UUID. A third optional parameter can be used to store extended information, such as data passed to the property page as it is displayed.

To add your own property page to a class, you edit either the Shell or the Admin property page attribute, depending on whether you want the default (shell) or administrator UI to be modified, and add in a line in the form of the preceding ones. It really is that simple. You can even modify the existing pages, if any, and resequence them to your liking.

Context Menus

When you right-click an object in ADUC, a context menu pops up. You can add your own entries to this context menu. Context menu items are held in the shellContextMenu attribute for the default UI and the adminContextMenu attribute for the admin UI in each displaySpecifier object. Items that should appear in both go into the contextMenu attribute.

The items that you add to the context menus can launch an application or create an instance of a COM object. The data takes the following form in the relevant attributes:

```
1,Extra &Data..., E:\MYPROG.EXE
2,&Extended Attributes...,C:\MYSCRIPT.VBS
3,{DB4909C2-6BEA-11D2-B1B6-00C04F9914BD}
```

Notice that the last item is a COM object. It is denoted by its GUID. The COM object must have been created to support the `IShellExtInit` and `IContextMenu` interfaces. Extra data can be passed to the COM object by including it as a third parameter on the line. The former two items are much more important to administrators. Here you can see that we added two extra items to the menu. These items are an executable program and a VBScript script. Any type of application is valid. The second parameter is the string you want to appear on the context menu. Use of an ampersand (&) character before a letter denotes that letter as the accelerator. Thus, when the menu is being displayed using the previous code example, typing "d" selects the first option, and "e" selects the second.

Being able to add scripts and programs to a context menu is a very powerful capability. Couple these scripts and programs with ADSI, and you have a way of extending the snap-ins Microsoft provides to deliver completely customized functionality based on your business or organizational needs. For example, let's say that you want to extend the schema and include a new, optional `myCorp-LanguagesSpoken` attribute for the `user` class. You can go to the `User-Display` object for the appropriate locale and modify the `contextMenu` attribute (so it is available to both users and administrators) to include an ADSI script that displays that attribute in a message box. The following code is all that is needed:

```
Option Explicit
Dim wshArgs, objUser
Set wshArgs = WScript.Arguments
Set objUser = GetObject(wshArgs(0))
MsgBox objUser.Get("myCorp-LanguagesSpoken"),,"Languages Spoken"
```

The script does nothing more than bind to the object's distinguished name that is passed in as an argument to the program, and print out the attribute in a `MsgBox` with an appropriate title, as shown in Figure 3-22.

In this figure, the `guest` user object was right-clicked, which popped up a context menu that includes Languages Spoken. You can see that it is actually the string "&Languages Spoken..." being displayed if you look at the text in the lower-left corner of the window. When we click the item or press the L key, a dialog box generated by the script is displayed on the screen. Normally the dialog box and the context menu would not be displayed together, but we have done so in this screen to show you the process.

You could also write a script or program that allowed you to modify the `mycorp-LanguagesSpoken` attribute and have it appear only on the administrator's context menus. Then you can use the ADUC tool to manage your users and this extra attribute, without ever needing to actually develop an entirely new interface if you don't want to.

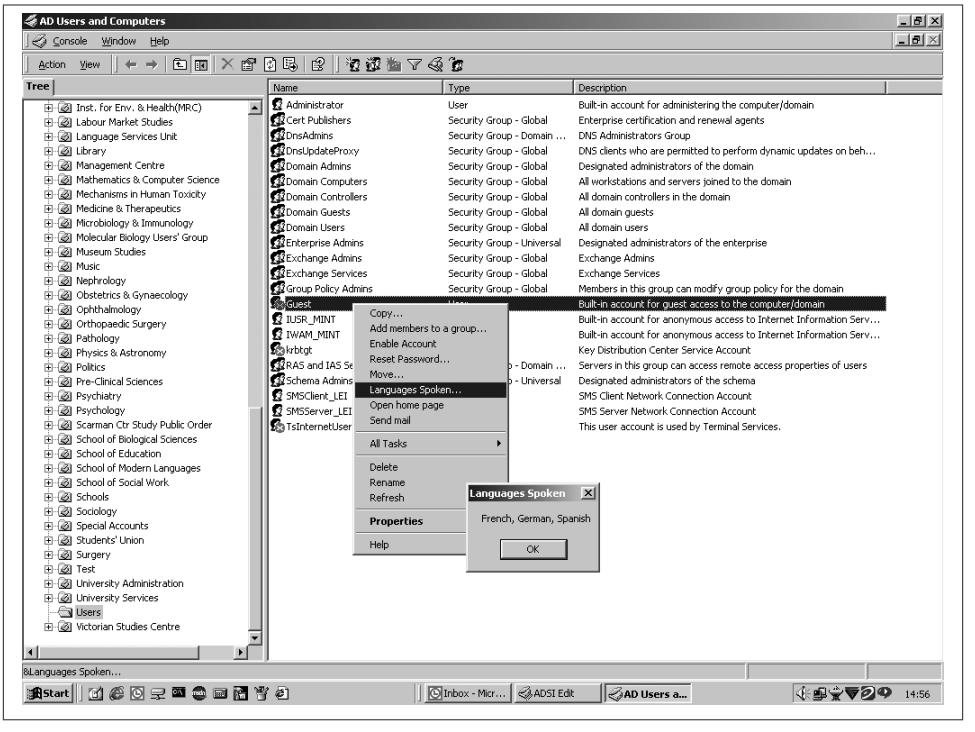


Figure 3-22. Viewing the languages spoken by a user

Icons

When you look at a container of objects in ADUC, it shows you an icon for each object that is appropriate to the specific object class for that object. The icons for organizational units look different than those for containers, users, and printers, for example. The icon can actually be used to represent different states of that object. For example, if you disable a user or computer object, the icon is changed to indicate that the object is disabled. All in all, 16 different state icons can be defined for any object class. The first three represent the states closed (the default state), open, and disabled; the last 13 are currently undefined and left for your own use.

To modify the icon for an object class, simply use the `iconPath` attribute to store multivalued data of the following form:

```
0, c:\windows\system32\myicon.ico
1, c:\windows\system32\myicons.dll, 0
2, c:\windows\system32\myicons.dll, 2
3, c:\windows\system32\myicons.dll, 7
```

This sets the first four icon values. Remember that 0 is closed, 1 is open, and 2 is disabled; 3 through 15 are undefined. The first line uses a proper icon file with an ICO extension and so doesn't need a third parameter. The last three use the first (0), third (2), and eighth (7) icons from *myicons.dll*, using an index for the set of icons held in the DLL, starting at 0. The icon path has to exist on the local machine for any client to properly display the icon. Remember to take that into account, since you may need to deploy the icon files to all clients in an enterprise if they are to display the icons properly.

Display Names

As shown earlier, you can alter the way that both class and attribute names appear within a GUI. If you want to change the class name, change the text in the `classDisplayName` property of the relevant `displaySpecifier` object. If you want to change what attribute names appear as, you need to modify the multivalued attribute `attributeDisplayNames`. Attribute values take the form of a comma-delimited string, as follows:

```
mobile,Mobile Number
physicalDeliveryOfficeName,Office Location
extensionAttribute1,My First Extended Attribute
```

The first value is the LDAP name corresponding to the attribute in the schema, and the second is the name that it is to be displayed as. Note that you shouldn't insert a space between the comma and the second value unless you want the name to be preceded by a space.

Leaf or Container

When you view objects in the ADUC, some display as containers and some display as leaf objects. Most objects that you are concerned with actually act as containers, even if you see them displayed as leaf objects. Take a printer on a computer, for example. If that printer is published as a `printQueue` object to Active Directory, the object is will appear as a leaf object within the computer object on which it is published. The computer object acts as a container for any print queues that it publishes, but by default, user, computer, and group objects are not displayed as containers. ADUC in fact has an option on the View menu to change this, called "View users, groups, and computers as containers." However, all objects will be treated in this fashion. Instead, you can modify a particular object's default setting by going to the `displaySpecifier` and changing the Boolean value of `treatAsLeaf` to `True` or `False` as required.

Object Creation Wizard

When you create a user, group, or organizational unit, ADUC presents a simple wizard to allow you to specify the relevant data for that object. It is possible for administrators to modify the default behavior in one of two ways: they can replace the existing wizard

entirely, if one exists, or just add extra pages to the wizard. Only one wizard can ever exist, so you either create a new one or modify the existing one. Let's say that you want to have the wizard ask for the value for the `myCorp-LanguagesSpoken` attribute for the `user` class. As the existing user-creation wizard does not allow data to be input for this attribute, you can replace the entire wizard with a new one of your own, or you can place a new page into the wizard to receive data on this attribute. With property pages, we need to create new wizards or creation wizard extensions (extra pages for existing wizards) as COM objects that support the `IDsAdminNewObjExt` interface. New wizards that replace the default wizards in use by the system are known as *primary extensions*, and they replace the core set of pages that would be used to create the object. Primary extensions support creation wizard extensions; you can define a primary extension for all users, for example, and later add a couple of extra pages using a creation wizard extension if you require.

If you are replacing the wizard entirely with a primary extension, modify the `creationWizard` attribute of the relevant `displaySpecifier` object to hold the GUID of the COM object. If you are just providing creation wizard extensions, you specify the order in which the pages should appear, followed by the UUID in the `createWizardExt` multi-valued attribute. The format is the same as for property pages.

Active Directory PowerShell Module

The Active Directory PowerShell module first appeared in Windows Server 2008 R2 and was enhanced in Windows Server 2012. While the module is currently in its second iteration, there are still some tasks that you cannot accomplish with it. ADAC uses the AD PowerShell module for all of its tasks, so, if you can do something in ADAC, that task can also be accomplished with the AD PowerShell module. In addition, as we discuss in Chapter 6, Windows Server 2012 introduces an extensive set of cmdlets for managing replication.

Throughout this book you'll find small examples of using the AD PowerShell module to accomplish various tasks, and we encourage you to read them and build on the benefits of using a scripting language such as PowerShell to be more efficient in your day-to-day Active Directory tasks. Microsoft has made a massive investment in PowerShell, especially in Windows Server 2012, and it is clearly the way of the future.

Even if you have never made a foray into scripting, we strongly encourage you to try it. The PowerShell History pane in ADAC makes discovering AD PowerShell cmdlets extremely easy, and PowerShell has matured substantially over the past several years and is now well documented on the Internet.

If you're looking for some resources related to learning PowerShell, we highly recommend *Windows PowerShell Cookbook, Second Edition* by Lee Holmes (O'Reilly).

Best Practices Analyzer

Keeping Active Directory healthy is a concern of any administrator, and the definition of a healthy domain controller or forest is a topic of much debate on the Internet. Historically, Microsoft has included a tool called *dcdiag* with Windows: the multitude of tests *dcdiag* supports can help an administrator proactively check the health of a domain controller, domain, or forest as well as troubleshoot problems when they arise.

The downfall of *dcdiag* is that the output is difficult to read, the list of switches required to test the intended targets is long, and there is typically no actionable guidance when a problem is identified. Starting with Windows Server 2008 R2, Microsoft set out to solve these problems by creating a Best Practices Analyzer (BPA) for Active Directory that is included as part of the operating system. Windows Server 2012 includes 41 tests that analyze many of the most common issues and misconfigurations that administrators typically run into.

To access the BPA, find Server Manager under the AD DS node on the left and scroll down to Best Practices Analyzer. To run a BPA scan, click Tasks→Start BPA Scan on the right. You can select one or more domain controllers to run the BPA scan on, as shown in Figure 3-23.

Once the scan completes, you'll see the results (Figure 3-24). Each result gives you the server exhibiting the issue, a severity level, the name of the issue and an overall category. If you refer to Figure 3-24, you'll see that when you select an issue, a description of the issue is shown below it along with a link to more information and remediation steps. For example, the issue selected in Figure 3-24 has the following guidance in Server Manager:

> **Problem:**
>
> Some organizational units (OUs) in this domain are not protected from accidental deletion.
>
> **Impact:**
>
> If all OUs in your Active Directory domains are not protected from accidental deletion, your Active Directory environment can experience disruptions that might be caused by accidental bulk deletion of objects.
>
> **Resolution:**
>
> Make sure that all OUs in this domain are protected from accidental deletion.

While the BPA can be run on demand in Server Manager, you can also invoke the BPA via Windows PowerShell. This could be useful for checking numerous servers at once, or for scheduling a regular BPA scan of a forest. To invoke the Active Directory BPA with Windows PowerShell, use the *Invoke-BpaModel* cmdlet:

```
Invoke-BpaModel "Microsoft/Windows/DirectoryServices"
```

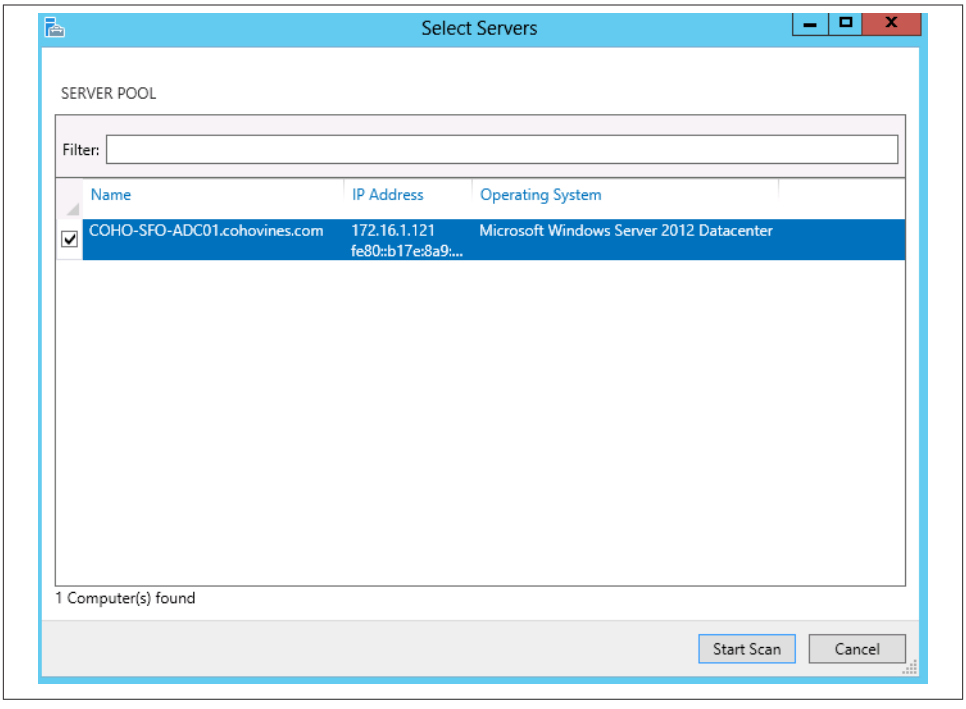


Figure 3-23. Selecting BPA target servers

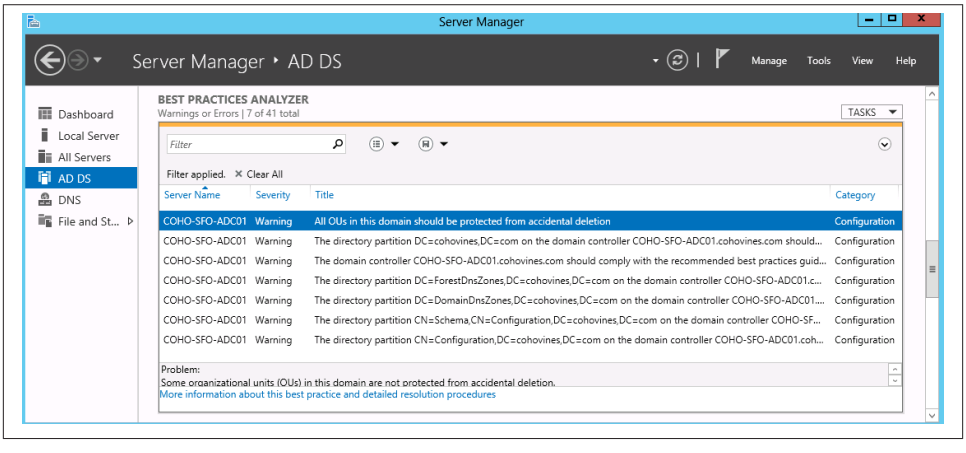


Figure 3-24. BPA results

Once the scan completes, use *Get-BpaResult* to view the results of the scan:

```
Get-BpaResult "Microsoft/Windows/DirectoryServices"
```

Using the BPA PowerShell cmdlets, you could schedule a task that checks for any warnings or errors in the results and sends an email alert if issues are found. In addition to Active Directory, many other components of Windows Server include a BPA. You can get a complete list of the available BPAs with the *Get-BpaModel* cmdlet:

```
Get-BpaModel | ft Id,Name
```

Active Directory-Based Machine Activation

When Microsoft released Windows Vista and Windows Server 2008, it changed the paradigm for installing licensed copies of Windows in corporate environments. Prior to this change, enterprises received special "volume license" installation media and keys that did not require online activation. This changed with the introduction of the key management server (KMS) that was activated with a special license key. Subsequently, machines on the network were directed to the KMS to activate.

Some of the challenges with the KMS model included the lack of authentication or security—any machine with network access could activate itself against the KMS. Additionally, enterprises were now forced to maintain additional servers and infrastructure just for licensing of Windows and Microsoft Office.

Windows 8 and Windows Server 2012 clients support activation from data stored in Active Directory. Since the activation data is stored in Active Directory, there is no longer any dependency on dedicated infrastructure and clients must also be joined to the domain (and thus implicitly authenticated) in order to activate their Windows installations.

You can publish activation information to the forest by using the Volume Activation Management Tool 3.0 (VAMT). The VAMT is included as part of the Windows Automated Deployment Kit (ADK) that is available for download from this link.

Summary

The goal of this chapter was to introduce the various management tools available to you as an Active Directory administrator, and some of the benefits and limitations of those tools. If you've been working with Active Directory for a while, chances are you've used Active Directory Users and Computers, but you might not have looked at the new Active Directory Administrative Center (ADAC). If you found ADAC in Windows Server 2008 R2 disappointing, check out the Windows Server 2012 version—it's been improved substantially.

In addition to the basic user, group, and computer management tools (ADUC and ADAC), we looked at LDP and ADSI Edit, two deeper tools that let you work with the directory on a much less abstract level. There's no GUI tool that can give you a more raw view into the directory than LDP. We also talked about the importance of harnessing a scripting language (such as Windows PowerShell) and the level of investment Microsoft has made in PowerShell capabilities for Active Directory.

Finally, we looked at a couple of important peripheral manageability investments in the form of the Best Practices Analyzer (BPA) that can analyze the health of your Active Directory environment, and AD-based machine activation that lets you keep domain-joined machines activated without a KMS server.

CHAPTER 4

Naming Contexts and Application Partitions

Due to the distributed nature of Active Directory, it is necessary to segregate data into partitions. If data partitions 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 all of the information within that domain. Information about other domains is not needed on those domain controllers. On the other hand, there is some Active Directory data that must be replicated to all domain controllers within a forest. 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 type of Active Directory data. The Configuration NC holds data pertaining to the configuration of the forest (or of forest-wide applications), such as the objects representing naming contexts, LDAP policies, sites, subnets, Microsoft Exchange, and so forth. 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.

Beginning with 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 for security principals. A major benefit of application partitions is that administrators can define which domain

controllers replicate the data contained within these partitions. Application partitions are not restricted by domain boundaries, as is the case with Domain NCs; they can exist on any domain controller running Windows Server 2003 or later in a forest, regardless of the domain the DC hosts.

You can retrieve a list of the naming contexts and application partitions a specific domain controller maintains by querying its *RootDSE* entry. You can view the *RootDSE* attributes by opening the LDP utility. To see how to view the *RootDSE* information, see the sidebar "Querying RootDSE with LDP" on page 64.

Querying RootDSE with LDP

ldp.exe is included with Windows Server 2008 and newer and is part of the Support Tools for Windows 2000, XP, and 2003. You can use LDP to view raw data from any LDAP server. We will leverage LDP throughout this book when appropriate. Unless otherwise noted, you should assume that the version of LDP we are using is the version that ships with Windows Server 2012.

The *RootDSE* is viewable by anonymously binding to any domain controller in the forest. The steps to use LDP to view this information are:

1. Launch LDP by running *ldp.exe*.
2. Click Connection→Connect, enter the name of a domain controller, and click OK. Figure 4-1 shows connecting to the domain controller *k8devdc01* on port 389 (the LDAP port).
3. LDP will connect to the domain controller and immediately display all of the anonymously viewable *RootDSE* output, as shown in Figure 4-2.

There is a great deal of data displayed in this screen, and we will not cover it all at this time. A good deal of the information will be covered throughout this book when it is relevant. The information of particular interest to us right now is the data corresponding to the attributes in Table 4-1.

We can see the distinguished name of the root domain in this forest (`DC=k8dev01,DC=brianlab,DC=local`), the DNs of the application partitions hosted on this domain controller (such as `DC=DomainDnsZones,DC=k8dev01,DC=brianlab,DC=local`), and so forth.

This output is particularly useful when writing scripts and programs that query Active Directory. By utilizing the *RootDSE* attributes, you can avoid hardcoding paths into your code.

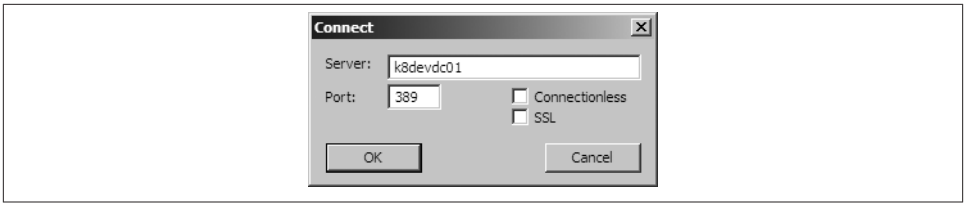


Figure 4-1. LDP connection dialog

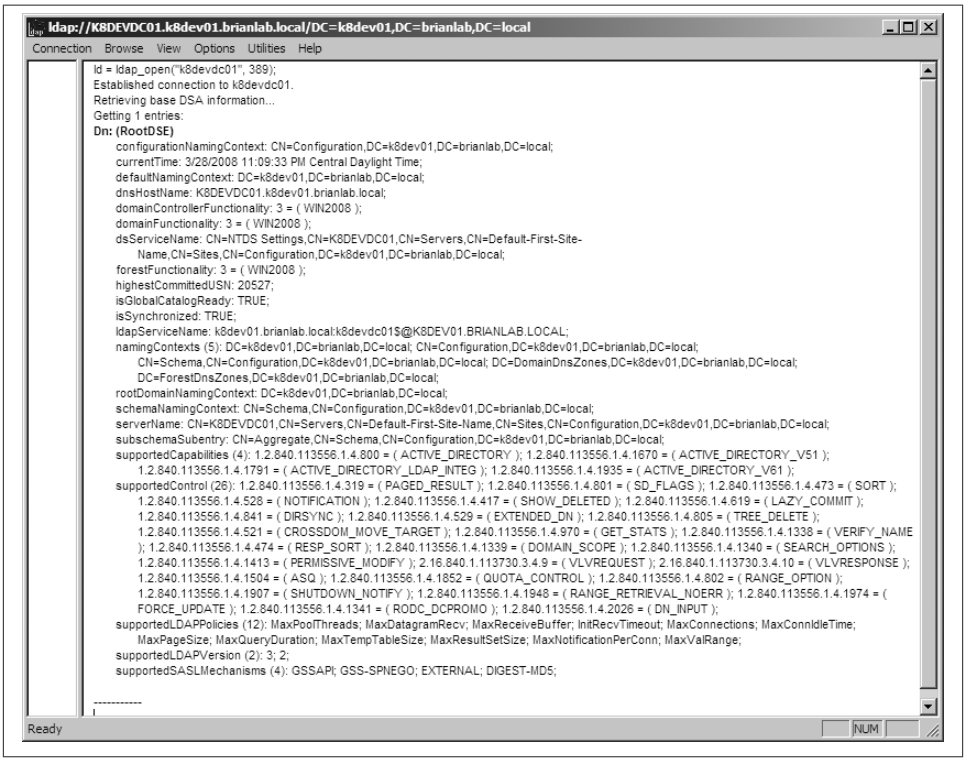


Figure 4-2. LDP RootDSE output on a Windows Server 2008 domain controller

Table 4-1. RootDSE attributes pertaining to naming contexts

Attribute Name	Description
`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 sample uses of them.

Domain Naming Context

Each Active Directory domain has a Domain naming context, which contains domain-specific data. The root of this NC is represented by a domain's distinguished name (DN) and is typically referred to as the *NC head*. 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 4-2 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. Alternatively, you can browse all of these containers with LDP or ADSI Edit.

To start ADSI Edit, go to Start→Run→**`adsiedit.msc`**.

Table 4-2. 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, Domain Users, and Account Operators.
`cn=Computers`	Default container for computer objects representing member servers and workstations. You can change the default container with the *redircmp.exe* utility.
`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. Orphaned objects are objects that were created in a container that was deleted from another domain controller within the same replication period.
`cn=Managed ServiceAccounts`	Container for managed service accounts. Managed service accounts are a special type of security principal that rotate their own password. For more information on managed service accounts, refer to Chapter 10.
`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.
`cn=Program Data`	Container for applications to store data instead of using a custom top-level container.
`cn=System`	Container for miscellaneous domain configuration objects. Examples include trust objects, DNS objects, and group policy objects.

Relative distinguished name	Description
cn=TPM Devices	Container for recovery information for Trusted Platform Module (TPM) keys.
cn=Users	Default container for user and group objects. You can change the default container with the *redirusr.exe* utility.

Configuration Naming Context

The Configuration NC is the primary repository for configuration information for a forest and is replicated to every domain controller in the forest. Additionally, every writable domain controller in the forest holds a *writable* copy of the Configuration NC. 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 4-3 contains a list of the default top-level containers found in the Configuration NC.

Table 4-3. Default top-level containers of the Configuration NC

Relative distinguished name	Description
cn=DisplaySpecifiers	Container that holds display specifier objects, which define various display formats for 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.
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.
cn=Partitions	Contains objects for each naming context, application partition, and external LDAP directory reference.
cn=Physical Locations	Contains location objects (`physicalLocation`), which can be associated with other objects to denote the location of the object.
cn=Services	Store of configuration information about services such as the File Replication Service, Exchange, and 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.

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. However, recall that the Schema NC is writable *only* on the domain controller holding the schema master FSMO role. The root of the Schema NC can be found in the Schema container, which is a child of the Configuration NC. 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 4-3 shows how the Schema and Configuration NCs are segregated in the ADSI Edit tool.

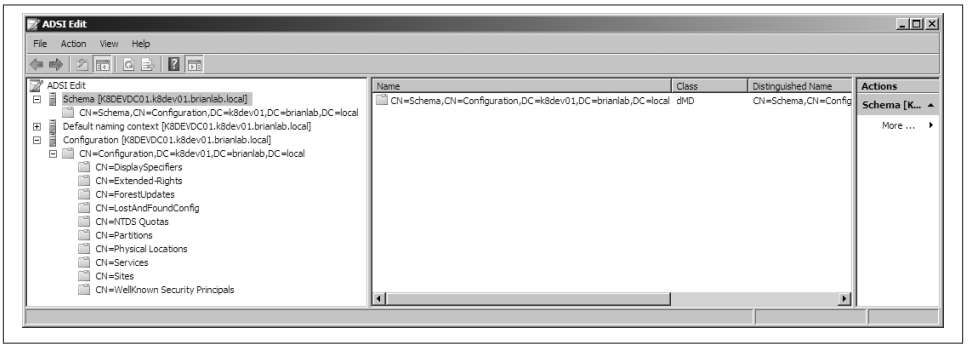


Figure 4-3. ADSI Edit view of the Configuration and Schema naming contexts

You may be wondering why the schema isn't just contained within the Configuration NC. As we discussed in Chapter 2, there is a schema master FSMO role that is the single master for updates to schema objects. This role is necessary due to the highly sensitive nature of the schema. Schema modifications need to be processed prior to any updates that utilize the schema. The mechanism to most easily guarantee this with the replication model AD uses is to put the schema into its own partition so it can replicate separately prior to other changes.

Unlike the Domain and Configuration NCs, the Schema NC does not maintain 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` instance that represents the abstract schema as defined in the LDAPv3 RFC.

Chapters 5 and 17 deal with the schema in more depth.

Application Partitions

Application partitions enable administrators to create areas in Active Directory to store data on specific domain controllers they choose, rather than on every DC in a domain or forest. You can define which domain controllers hold a copy of each application partition, which is known as a *replica*. There is no limitation based on domain or site membership, which means that you can configure any domain controller running Windows Server 2003 or later within 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 Service Location (SRV) records (explained in more detail in Chapter 8), 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 regarding application partitions:

- Application partitions cannot contain security principals, which most notably includes user, `inetOrgPerson`, 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 from objects contained in domains, which can be moved between domains.

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, or they can be nested under other application partitions (for example, `dc=sales,dc=apps,dc=mycorp,dc=com`), or be part of a new domain tree (for example, `dc=apps,dc=local`). For more information on creating and managing application partitions, refer to the sidebar "Creating Application Partitions".

Creating Application Partitions

Application partitions are commonly managed with the *ntdsutil* utility. With *ntdsutil*, you can manage application partitions in Active Directory and AD LDS. The types of operations you can perform include creating and deleting new application partitions, and adding and removing domain controllers and AD LDS instances to and from the list of replicas for an application partition.

In this example, we will create an application partition under the *cohovines.com* domain called `MyPart`. This domain is at the Windows Server 2008 functional level. We will enable the application partition to replicate to domain controllers *dc01* and *dc02*.

Here are the steps required to create the application partition:

1. From a command prompt, launch *ntdsutil*.
2. Enter partition management mode by entering **`partition management`**.
3. Enter the distinguished name of the application partition and a server to create the partition on:

```
create nc "dc=MyAppPart,dc=cohovines,dc=com" dc01.cohovines.com
```

4. Enter **`quit`** to exit the `partition management` menu, and then enter **`quit`** again to exit *ntdsutil*.

Here's how to add the *dc02* replica:

1. From a command prompt, launch *ntdsutil*.
2. Enter partition management mode by entering **`partition management`**.
3. Enter **`connections`** to move to the `connections` submenu.
4. Enter **`connect to server dc02`**. You must always connect to the server to which you will be adding the replica.
5. Enter **`quit`** to return to the `partition management` menu.
6. Enter the following NC replica to add the replica to *dc02*:

```
add nc replica "dc=MyAppPart,dc=cohovines,dc=com" dc02.cohovines.com
```

7. Enter **`quit`** to exit the `partition management` menu, and then enter **`quit`** again to exit *ntdsutil*.

On Windows Server 2003, you should replace the *partition management* command with the *domain management* command. The rest of the syntax is identical.

Application partitions tend to store dynamic data—that is, data that has a limited lifespan (see the section on "Storing Dynamic Data" on page 71 for more on this). Dynamic data from network services such as DNS and the Dynamic Host Configuration Protocol (DHCP) can reside in an application partition in AD. This allows uniformity of access from applications via a single methodology, which in turn enables developers to write to a special area only available on specific servers rather than into a domain partition that is replicated to every DC. In fact, application partitions could allow multiple versions of COM+ applications to be installed and configured on the same computer, resulting in more cost-effective management of server applications.

The availability of Active Directory Lightweight Directory Services (AD LDS) has given administrators another option for storing directory data outside of the normal domain-naming contexts while still using Windows security and authentication. Instead of putting application data in an application partition, you can place that data in a dedicated AD LDS instance. This allows you to offload administrative control of that information to application owners or other administrators, as well as lessening the chance of an application negatively impacting a domain controller's primary NOS function. We discuss AD LDS specifics in Chapter 20.

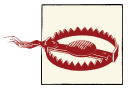
Storing Dynamic Data

Although 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 they will exist before being automatically deleted by Active Directory. Dynamic objects typically have a fairly short lifespan (i.e., days or less). An example use of dynamic objects is an ecommerce website that needs to store user session information temporarily. Because 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 TTL that is set for dynamic objects is 1 day, but can be configured to be as short as 15 minutes.

The default TTL and minimum TTL can be modified by changing the `DynamicObjectDefaultTTLSeconds` and `DynamicObjectMinTTLSeconds` values in the `ms-DS-Other-Settings` attribute of the `CN=DirectoryService,CN=WindowsNT,CN=Services,CN=Configuration,DC=...` object.

To create a dynamic object, you simply have to add `dynamicObject` to the `objectClass` attribute when creating the object. Microsoft has specifically disabled the ability

to add this to existing objects for safety reasons. This is why you cannot convert existing static objects into dynamic objects. The `entryTTL` attribute can also be set at creation time to set the TTL to something other than the one-day default. To prevent a dynamic object from being automatically deleted, you can "refresh" the object by resetting the `entryTTL` attribute for the object to a new TTL value (time specified in seconds).



Dynamic objects do not get tombstoned like normal objects when they are deleted; they are just removed from the directory. A tombstone is not needed since the TTL mechanism allows them to be immediately removed from all domain controllers simultaneously. For more information on tombstones, see the section "Preserving attributes in a tombstone" on page 89 in Chapter 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 provide a way for administrators to define their own groupings 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 allow you to create objects that have a TTL value. After the TTL expires, Active Directory automatically deletes the object.

CHAPTER 5

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, or it can be null. A null value has an implementation-specific meaning; it could mean `True` or `False` depending on the application using the 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. Each new release of Active Directory has included updates to the default schema. For background information on schema versions, see the sidebar "Schema Versions" on page 73, next. Additionally, the Active Directory schema was designed to be extensible, so that administrators could add classes or attributes they deemed 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 17. In this chapter, we're concerned only with the fundamentals of the schema.

Schema Versions

Each time Microsoft releases an update to the default Active Directory schema, it updates the `schemaVersion` attribute in Active Directory. To date, there have been four versions of the default Active Directory schema release, as that are outlined in Table 5-1. This version is not updated when administrators make their own custom changes, so while the version number tells you the minimum level of the schema from the OS perspective, it does not attest to whether there have been additional specific modifications as well.

You can easily query the schema version with the *adfind* command-line tool. Use this command to do so:

```
adfind -schema -s base objectVersion
```

Table 5-1. Active Directory default schema versions

Schema version	Release
13	Windows 2000
30	Windows Server 2003
31	Windows Server 2003 R2
39	Windows Server 2008 Beta Schema[a]
44	Windows Server 2008
47	Windows Server 2008 R2
56	Windows Server 2012

[a] Reference this link for more information about this schema version.

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 *contoso.com* forest would be `cn=schema,cn=Configuration,dc=contoso,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 (though in reality, all the schema objects are stored directly in the Schema container).

The Active Directory Schema MMC snap-in is not fully enabled by default. In order to enable the schema management snap-in on a domain controller, you must first register the DLL it depends on. To do that, run this command: *regsvr32 schmmgmt.dll*.

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` (ClassSchema) 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 `lDAP DisplayName` attribute of each object is normally made by capitalizing the first letter of

each word of the Common-Name, and 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.

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. If the class you are adding will have custom attributes that are required to be populated when new instances of that class are created, you must define the `attributeSchema` objects first. If you just want to add a new attribute to an existing class, you must 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 infrastructure but also throughout the world.

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 Telecommunication Union) organizations in 1988. To properly understand how the Active Directory schema works, you really need to understand some of 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 its 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:

- The first part indicates the unique path to the branch holding the object in the X.500 tree-like structure.
- The second part uniquely indicates the object 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
```

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`.

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

Each branch within an OID number also corresponds to a name. This means that the dotted notation 1.3.6.1.4.1, for example, is equivalent to *iso.org.dod.internet.private.enterprise*. As the names are of no relevance to us with Active Directory, we don't cover them in this book.

This notation continues today and is used in the Active Directory schema. If you wish to create a schema object, you need to obtain a unique OID branch for your organization. Using this as your root, you can then create further branches and leaf nodes within the root, as your organization requires.

The Internet Assigned Numbers Authority (IANA) maintains the main set of root branches and defines itself as "the central coordinator for the assignment of unique parameter values for Internet protocols." The IANA says of its mission:

> The IANA is chartered by the Internet Society (ISOC) and the Federal Network Council (FNC) to act as the clearinghouse to assign and coordinate the use of numerous Internet protocol parameters. The Internet protocol suite, as defined by the Internet Engineering Task Force (IETF) and its steering group (the IESG), contains numerous parameters, such as Internet addresses, domain names, autonomous system numbers (used in some routing protocols), protocol numbers, port numbers, management information base object identifiers, including private enterprise numbers, and many others. The common use of the Internet protocols by the Internet community requires that the particular values used in these parameter fields be assigned uniquely. It is the task of the IANA to make those unique assignments as requested and to maintain a registry of the currently assigned values. The IANA is located at and operated by the Information Sciences Institute (ISI) of the University of Southern California (USC).

You can find the IANA website at this link.

You can request an OID namespace—i.e., a root OID number from which you can create your own branches—directly from the IANA if you like. These numbers are known as Enterprise Numbers. The entire list of Enterprise Numbers can be found at iana.org. This list of numbers is updated every time a new one is added.

At the top of the file, you can see that the root that the IANA uses is 1.3.6.1.4.1. If you look down the list, you will see that Microsoft has been allocated branch 311 of that part of the tree, so Microsoft's OID namespace is 1.3.6.1.4.1.311. As each number also has a contact email address alongside it in the list, you can search through the file for any member of your organization that has already been allocated a number. It is likely that large organizations that already have an X.500 directory or that have developed SNMP management information bases (MIBs) will have obtained an OID.

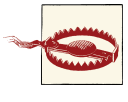
In addition to Enterprise Numbers, country-specific OIDs can be purchased. An organization's Enterprise Number registration has no bearing on whether it has obtained a country-based OID namespace to use. If you don't see the company listed in the Enterprise Numbers list, don't be fooled; the organization could still have a number.

For example, Microsoft has been issued the Enterprise Number 1.3.6.1.4.1.311, yet all of its new schema classes use a US-issued OID namespace of 1.2.840.113556 as their root. The 1.2.840 part is uniquely allotted to the United States. In other words, Microsoft has obtained two OID namespaces that it can use but is choosing to use only the US-issued namespace.

If you want to obtain an Enterprise Number, fill in the online form at *http://pen.iana.org/pen/PenApplication.page*. If this URL changes, you can navigate to it from the main IANA web page.

Microsoft used to issue unique OID namespaces to customers on request; however, it no longer does this. Instead, Microsoft provides a script that will generate a statistically unique OID branch each time it is run. This script is available from *http://go.microsoft.com/fwlink/?LinkId=100725*. While this script is a viable solution, best practice dictates that you should obtain a globally unique OID namespace and issue OIDs from there instead.

Once an organization has an OID namespace, it can add unique branches and leaves in any manner desired under the root. For example, you could decide to have no branches underneath and just give any new object an incrementing integer starting from 1 underneath the root. Alternatively, you could decide to make a series of numbered branches starting from 1, each corresponding to a certain set of classes or attributes. Thus, the fifth object under the third branch would have an OID ending in 3.5. Figure 5-1 shows one sample scenario for how an organization decided to structure its OID namespace.



The range of values in any part of an OID namespace for the Active Directory schema goes from 1 to 268,435,455—i.e., from 2^0 through $2^{28} - 1$.

This limitation has caused issues with schema extensions for some companies in Australia. Australia has the OID 1.2.36, and according to the Australia Standards document MP-75, companies may use their Australian Company Number, or ACN (excluding leading zeros), to formulate their OIDs without needing to request one. Unfortunately the ACN is nine digits, so it could easily exceed the limitation listed above.

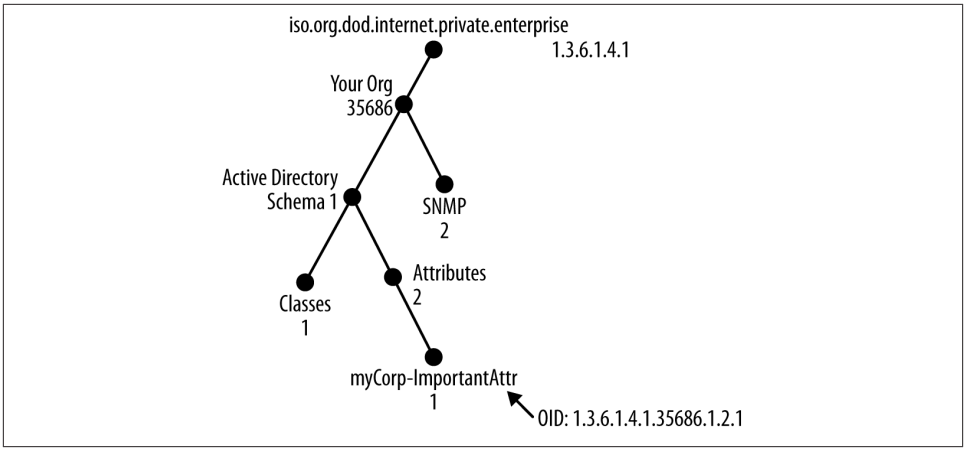


Figure 5-1. Sample OID namespace organizational tree

To reinforce this point, let's look at a couple of examples directly from the Active Directory schema. If you open the Active Directory Schema snap-in, you can look at the schema class OIDs very easily. Navigating through the classes when we open the property page for the `printQueue` class, we get Figure 5-2. You can see that the unique OID is 1.2.840.113556.1.5.23. This tells us that the number is a defined part of Microsoft's object class hierarchy.

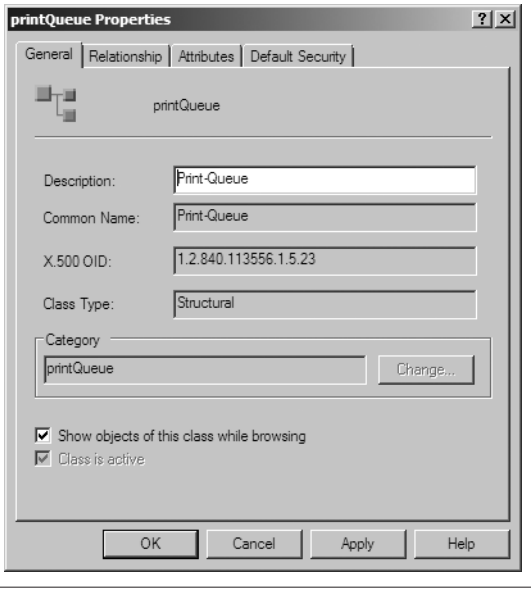


Figure 5-2. printQueue schema class properties

Figure 5-3 shows the property page for the `organizationalPerson` class. Here, you can see that the unique OID 2.5.6.7 is very different, because within the original X.500 standard, a set of original classes was defined. One was `organizationalPerson`, and this is a copy of that class. Microsoft included the entire set of base X.500 classes within Active Directory.

The OID numbering notation has nothing to do with inheritance. Numbering a set of objects a certain way does nothing other than create a structure for you to reference the objects; it does not indicate how objects inherit from one another.

Let's dissect an example attribute and class to see what they contain. With that information, you will be able to see what is required when you create a new schema object.

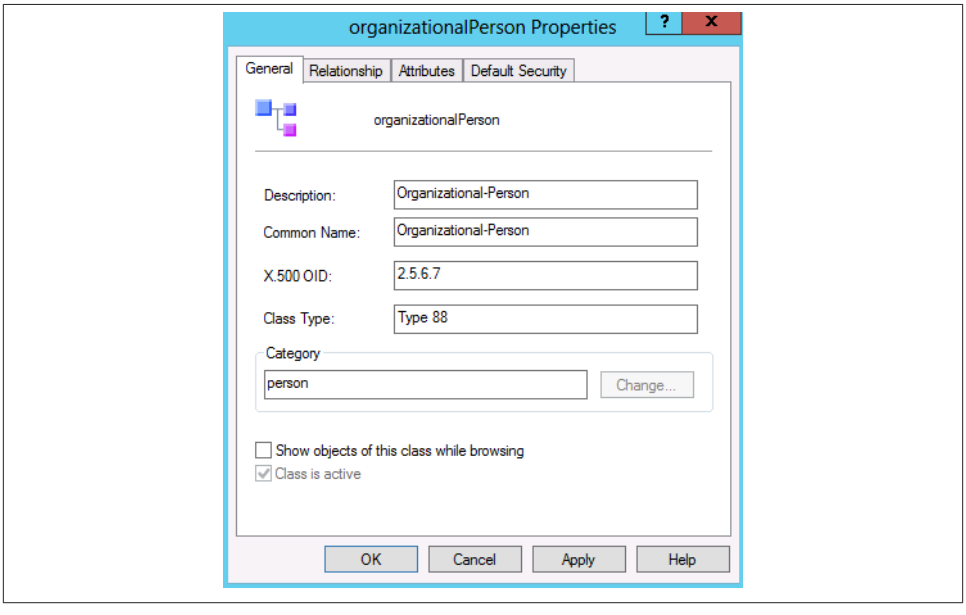


Figure 5-3. organizationalPerson schema class properties

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 relevant here. All of the attributes of the attributeSchema class are documented at this site.

Dissecting an Example Active Directory Attribute

The userPrincipalName (UPN) attribute is used on user objects to provide a unique method of identifying each user across a forest. Users can log =on to a workstation in any domain in the forest using the UPN if they so desire. The UPN attribute, in fact, accepts valid RFC 2822 (email) addresses, so the UPN for user *tpood* in the *europe.contoso.com* domain could be *tpood@contoso.com* or *tpood@europe.contoso.com*, or even *tpood@logon.local*. In fact, any UPN suffix, such as *@contoso.com*, can be used in a forest. The only requirement is that the UPN value for a user is unique across all users in a forest.

Active Directory does not enforce the uniqueness of a UPN when it is set. If two different users in the same forest are assigned the same UPN, neither will be able to log on using the UPN. When duplicate UPNs are detected, domain controllers will log an event from source Key Distribution Center (KDC) with event ID 11. Many large organizations implement scripts or other tools to scan their directories on a regular basis to check for duplicate UPNs.

To dissect the attribute, we need to find out what values have been set for it. Table 5-2 shows a subset of the values of attributes that have been set for the userPrincipalName attributeSchema instance.

Table 5-2. userPrincipalName's attributes

Attribute lDAPDisplayName	Attribute syntax	Attribute value
adminDescription	CASE_IGNORE_STRING	User-Principal-Name
adminDisplayName	CASE_IGNORE_STRING	User-Principal-Name
attributeID	CASE_IGNORE_STRING	1.2.840.113556.1.4.656
attributeSyntax	CASE_IGNORE_STRING	2.5.5.12
cn	CASE_IGNORE_STRING	User-Principal-Name
isMemberOfPartialAttributeSet	BOOLEAN	True
isSingleValued	BOOLEAN	True
lDAPDisplayName	CASE_IGNORE_STRING	userPrincipalName

Attribute lDAPDisplayName	Attribute syntax	Attribute value
name	CASE_IG NORE_STRING	User-Principal-Name
objectCategory	DN_STRING	cn=Attribute-Schema, cn=Schema,a=Con figuration, dc=mycorp,dc=com
objectClass	CASE_IG NORE_STRING	top; attributeSchema (two values of a multivalued attribute)
oMSyntax	INTEGER	64
searchFlags	INTEGER	1 (indexed)
showInAdvancedViewOnly	BOOLEAN	True
systemFlags	INTEGER	18 (category 1 attribute, replicated to GC)
systemOnly	BOOLEAN	False

We can see that the name of the attribute is `User-Principal-Name` (`adminDescription, adminDisplayName, cn, name`), that it is an instance of the `attributeSchema` class (`objectCategory` and `objectClass`), that it inherits attributes from both `top` and `attributeSchema` (`objectClass`), and that the UPN attribute is not visible to casual browsing (`showInAdvancedViewOnly`).

The `userPrincipalName` attributes show the following:

- It is to be stored in the GC (`isMemberOfPartialAttributeSet` and `systemFlags`).
- It is to be indexed (`searchFlags`).
- It has an OID of 1.2.840.113556.1.4.656 (`attributeID`).
- We should use `userPrincipalName` (`lDAPDisplayName`) when binding to it with ADSI.
- Instances can be created by anyone (`systemOnly`).
- It stores single (`isSingleValued`) Unicode strings (`attributeSyntax` and `oMSyntax`).

In Figure 5-4, you can see many of the values for the UPN attribute. We have indicated which attributes are changed by checking or unchecking each checkbox.

Attribute Properties

There are several properties of attributes that have a significant and varied impact on attribute use and functionality. Here we give a little more detailed information on a few attributes that you need to understand when modifying the schema.

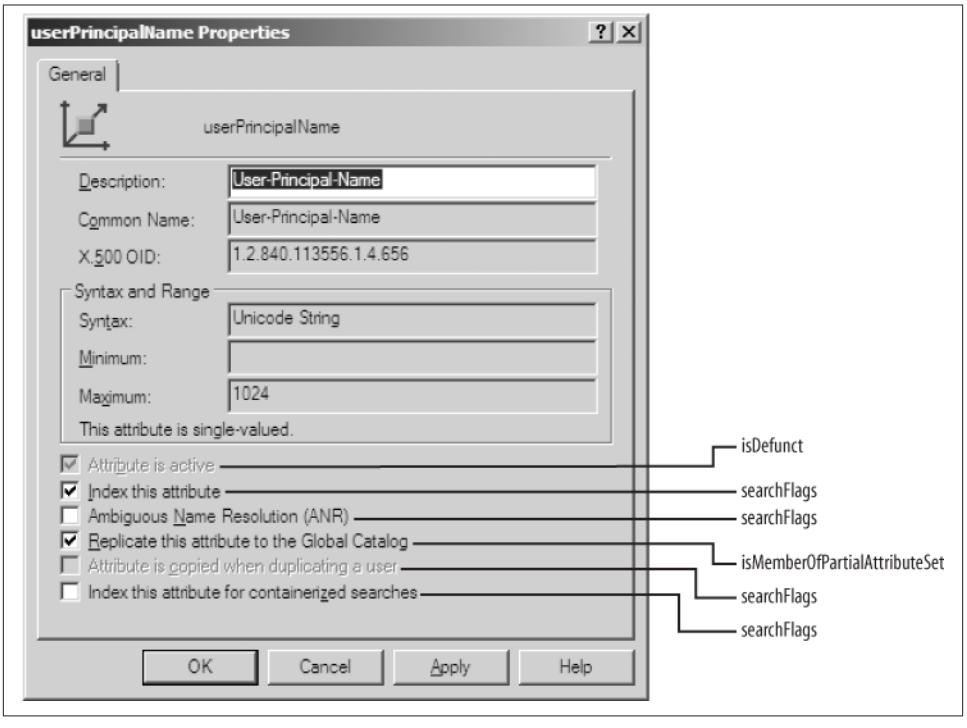


Figure 5-4. UserPrincipalName attribute settings

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 22 syntaxes, you must specify two 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 5-3. More than one syntax has the same OID, which may seem strange; to uniquely distinguish between different syntaxes, you thus need a second identifier. This is the result of Microsoft requiring some syntaxes that X.500 did not provide. Table 5-3 shows the 22 expanded syntaxes, including the name of each syntax with alternate names following in parentheses.

Table 5-3. Syntax definitions

Syntax	OID	OM syntax	Description
Address	2.5.5.13	127	Used internally by the system.
Boolean	2.5.5.8	1	True or false.
Case-insensitive string	2.5.5.4	20	A string that does not differentiate between uppercase and lowercase.
Case-sensitive string	2.5.5.3	27	A string that differentiates between uppercase and lowercase.
Distinguished name	2.5.5.1	127	The fully qualified domain name (FQDN) of an object in Active Directory.
DN-Binary	2.5.5.7	127	Octet string with binary value and DN. Format: *B:<char count>:<binary value>:<object DN>*.
DN-String	2.5.5.14	127	Octet string with string value and DN. Format: *S:<char count>:<string value>:<object DN>*.
Generalized-Time	2.5.5.11	24	ASN1.1 time format, e.g., 20040625234417.0Z.
Integer (enumeration)	2.5.5.9	10	A 32-bit number.
Integer (integer)	2.5.5.9	2	A 32-bit number.
Large integer	2.5.5.16	65	A 64-bit number.
NT security descriptor	2.5.5.15	66	A security descriptor (SD).
Numeric string	2.5.5.6	18	A string of digits.
Object ID	2.5.5.2	6	An OID.
Octet string (Octet-String)	2.5.5.10	4	A byte string.
Print case string (IA5-String)	2.5.5.5	22	A normal printable string.
Print case string (Printable-String)	2.5.5.5	19	A normal printable string.
Replica-Link	2.5.5.10	127	Replication information.
SID	2.5.5.17	4	A security identifier (SID).
Undefined	2.5.5.0	N/A	Not a valid syntax.
Unicode	2.5.5.12	64	A wide string.
UTC-Time	2.5.5.11	23	The number of seconds elapsed since 1 January 1970.

Most of these are standard programming types. If you're not sure which syntax to use, take a look at some preexisting attributes and see if you can find an appropriate syntax for the attribute you wish to create. For example, the `userPrincipalName` attribute has an `attributeSyntax` of 2.5.5.12 and an `oMSyntax` of 64, so it must contain Unicode strings.

systemFlags

The `systemFlags` attribute is an often overlooked but important attribute. It is a bit mask that represents how the attribute should be handled. For more information on bit masks, see the upcoming sidebar "How to Work with Bit Masks" on page 85. New bit values can be defined any time that Microsoft updates the directory service binaries. The `systemFlags` attribute is configured both on schema definitions of attributes and classes and on any instantiated object throughout the forest. This can be confusing, but the various bits in the attribute can mean various things depending on the object to which the attribute applies. Table 5-4 lists only the values for `systemFlags` on `attributeSchema` and `classSchema` objects. A complete listing of valid `systemFlags` settings is available at this link.

Table 5-4. System flag values for class and attribute objects

Value	Description
1 (0x0001)	Attribute is not replicated.
2 (0x0002)	Attribute will be replicated to the Global Catalog. This value should only be set by Microsoft; do not use it. Instead, use the `isMemberOfPartialAttributeSet` attribute for adding attributes to the partial attribute set.
4 (0x0004)	Attribute is constructed, not stored in the database. This should only be set by Microsoft; do not use it.
16 (0x0010)	Category 1 attribute or class. Category 1 objects are classes and attributes included in the base schema with the system. Note: not all classes and attributes in the base schema are marked Category 1.
33554432 (0x02000000)	The object is not moved to the Deleted Objects container when it is deleted. Instead, the tombstone remains in the original container.
134217728 (0x08000000)	The schema object cannot be renamed.
268435456 (0x10000000)	For objects in the configuration partition, if this flag is set, the object can be moved with restrictions; otherwise, the object cannot be moved. By default, this flag is not set on new objects created under the configuration partition. This flag can only be set during object creation.
536870912 (0x20000000)	For objects in the configuration partition, if this flag is set, the object can be moved; otherwise, the object cannot be moved. By default, this flag is not set on new objects created under the configuration partition. This flag can only be set during object creation.
1073741824 (0x40000000)	For objects in the configuration partition, if this flag is set, the object can be renamed; otherwise, the object cannot be renamed. By default, this flag is not set on new objects created under the configuration partition. This flag can only be set during object creation.
2147483648 (0x80000000)	The object cannot be deleted.

How to Work with Bit Masks

Masks are a fundamental concept in computer science, and perhaps the most common type of mask is the bit mask. A fair number of attributes in Active Directory are actually bit masks. Bit masks are series of binary values that often represent a series of settings. Bit masks can be confusing to administrators since they are sometimes displayed as a decimal number, whereas the actual data is a series of bits (binary data).

Let's examine the following bit mask, which describes a very limited set of animals:

Description	Alive	Mammal	Bird	Cat	Dog	Parrot	Tiger	Monkey
Bit number	0	1	2	3	4	5	6	7
Bit	0	1	0	0	0	0	0	1
Decimal representation	1	2	4	8	16	32	64	128

In this case, we set the necessary bits to describe a monkey that is not currently living. The binary representation of this mask is `01000001`, and in decimal, it's `130` (2 + 128). Each of the bits represents a distinct characteristic of an animal. In order to fully describe our monkey, it was necessary to set two bits.

Now, we probably wouldn't even have a mask representing the monkey in this state if the monkey wasn't in the production process, so once the monkey is born, we'll need to update the mask to reflect its "Alive" state (a.k.a. bit one). In order to do this, you need to do a binary OR operation, which is equivalent to addition:

```
01000001 OR 10000000 = 11000001 (01000001 + 10000000)
```

You can use the scientific view in the Windows calculator to perform binary arithmetic operations. The new decimal representation is `131`.

This may seem simple; however, it is a very common error for administrators to modify an attribute that is a bit mask by replacing the decimal value shown in the administrative tool with another decimal value. When you do this, data can be lost or added inadvertently. Consider, for example, if you were updating the monkey's mask to represent its alive status and you copied the decimal value from a live parrot—you would replace the live monkey's mask with `37`, which then describes the monkey as a parrot.

The moral of the story here is that you should always treat bit masks as binary data and alter them accordingly.

Constructed attributes

Most attributes are directly stored in the Active Directory database. Constructed attributes are the exception, and they are handled by the directory service in order to offer special functionality. This functionality can range from telling you approximately how

many objects are contained directly under a container type object (`msDS-Approx-Immed-Subordinates`) to telling you the types of objects that can be instantiated under a given object (`possibleInferiors`) to telling you which attributes you have write access to on a given object (`allowedAttributesEffective`), and many other things. These attributes, because they are special, have some rules you should be aware of:

- Constructed attributes are not replicated. They are constructed by each directory instance separately.
- Constructed attributes cannot be used in server-side sorting.
- Constructed attributes generally cannot be used for queries. The attribute `aNR` is an exception here as it is used for constructing the special ANR queries. ANR is covered in the section "Ambiguous name resolution" on page 89.
- In some cases, a base scope query may be required to retrieve certain constructed attributes due to the computational cost of constructing the attribute; e.g., `tokenGroups` can only be returned with a base scope query.

Category 1 objects

Category 1 objects are a subset of the attributes and classes that come with AD LDS or Active Directory. They are marked with a special bit flag so that Microsoft can track them and protect them from certain types of modifications.

schemaFlagsEx

The `schemaFlagsEx` attribute is an attribute that has existed since Windows 2000 but was not put into use until Windows Server 2008. It is designed to hold flags that further define the properties of an attribute. There is currently only one flag implemented in this bit mask, as outlined in Table 5-5.

Table 5-5. schemaFlagsEx values

Bit number	Value	Description
1	1 (0x1)	Marks an attribute as critical. Critical attributes cannot be added to the filtered attribute set, regardless of the value of the tenth bit of the `searchFlags` attribute.

searchFlags

The `searchFlags` attribute is another bit mask that is best known as the attribute used to control indexing, but it is a little more involved than that. As indicated by the name, `searchFlags` is similar to `systemFlags` in that it is a series of bits representing how the attribute should be handled. Unlike `systemFlags`, `searchFlags` is only set on schema attribute definitions. See Table 5-6 for all of the values.

Table 5-6. SearchFlags bits

Bit number	Value	Description
1	1 (0x0001)	Create an index for the attribute. All other index-based flags require this flag to be enabled as well. Marking linked attributes to be indexed has no effect.
2	2 (0x0002)	Create an index for the attribute in each container. This is only useful for one-level LDAP queries.
3	4 (0x0004)	Add an attribute to ambiguous name resolution (ANR) set. ANR queries are primarily used for Exchange and other address book tools. ANR attributes must be indexed and must be in either UNICODE or Teletex string attribute syntax. Adding attributes to this set can have performance implications in Microsoft Exchange.
4	8 (0x0008)	Preserve this attribute in a tombstone object. This flag controls what attributes are kept when an object is deleted.
5	16 (0x0010)	Copy this value when the object is copied. This flag doesn't do anything in Active Directory; tools such as Active Directory Users and Computers that copy objects can look at this flag to determine what attributes should be copied.
6	32 (0x0020)	Create a tuple index. Tuple indexing is useful for medial searches. A medial search has a wildcard at the beginning or in the middle of the search string. For example, the medial search (`drink=*coke`) would match Cherry Coke, Diet Coke, etc.
7	64 (0x0040)	Create a subtree index. This index is designed to increase the performance of virtual list view (VLV) queries.
8	128 (0x0080)	Mark the attribute as confidential. Only users with both Read Property *and* Control Access rights to the attribute so marked can view it when it is so marked.
9	256 (0x0100)	Never audit changes to this attribute. This flag was new in Windows Server 2008. Auditing is covered in Chapter 16.
10	512 (0x0200)	Include this attribute in the RODC filtered attribute set. RODCs and the filtered attribute set are covered in Chapter 9.

Indexed attributes

Attribute indexing is available to boost query performance. When an attribute is indexed, the values are placed in a special table in a sorted order so that a query using the attribute can be completed by looking at a subset of all the information in the directory. The type of index created can be modified by additional bit flags configured in the `searchFlags` attribute. There are several points to know about indexes:

- A query that contains bitwise operations on an indexed attribute diminishes the usefulness of the index. A bitwise operation can't be directly looked up in the index table, and the entire set of values in the index will have to be enumerated and tested. Bitwise queries are queries that query a bit mask (for example, the `systemFlags` or `userAccountControl` attributes).
- A query that contains a NOT of an indexed attribute negates the use of the index for that portion of the query. A NOT of an attribute requires enumerating all objects

in the search scope to determine which objects don't have the attribute or which objects have permissions applied that disallow the trustee to view the attribute value.

- Linked attributes are implicitly indexed. If you modify the flag, it will have no effect due to the implicit indexing behavior.
- It is often assumed that indexes cannot be built or do not work well for attributes with multiple values or non-unique values. This is incorrect. In the early days of the original Active Directory beta, there was concern about multiple values and non-unique values, but the issues surrounding them were addressed. This topic is most often raised in regard to the `objectClass` attribute and is stated as the reason why Microsoft didn't index the attribute by default prior to Windows Server 2008.

  If you have installed the Exchange Server 2007 schema extensions in your forest, `objectClass` is indexed as part of this schema extension.

  Windows Server 2008 and newer domain controllers implement a special behavior that indexes `objectClass` by default regardless of the `searchFlags` setting in Active Directory. Note that this is considered a special behavior because the attribute will not be indexed on Windows 2000 or Windows Server 2003 domain controllers once you import the Windows Server 2008 or newer schema extensions; however, the attribute will be indexed on any up-level domain controllers. This could cause performance deltas in applications that randomly select domain controllers for use. You may consider indexing `objectClass` when you apply the Windows Server 2008 or newer schema updates so that you will have a consistent LDAP query experience across your enterprise once newer domain controllers are introduced.

While indexing attributes can very frequently improve the performance of LDAP queries, it is important to realize that indexes also consume disk space. Adding an index to an attribute that is populated across a large percentage of directory objects may consume a substantial amount of disk space.

Domain controller performance will also be impacted while indexes are being generated. Index data is not replicated, so every domain controller in the forest must build its own copy of an index when it detects a new attribute index must be created. The speed at which an index is created is dependent on how much data must be indexed and also the hardware on which the domain controller is running.

Windows Server 2012 DCs support the notion of *deferred indexing*. When deferred indexing is enabled, rather than beginning the index generation process immediately, the DC will not generate an index until it is either rebooted or an `UpdateSchemaNow` LDAP request is made. In order to enable deferred indexing, you must modify the forest's `dsHeuristics` value such that the 19th byte is equal to `1`. For more information on editing the `dsHeuristics` attribute, refer to Chapter 16.

Ambiguous name resolution

Ambiguous name resolution (ANR) is used for address book lookups. It allows a single small query to be expanded into searching as many fields as the administrator would like searched, so that users can enter a single piece of information and hopefully find all possible "hits" on the value they are interested in. When an ANR query such as `(anr=Joe Richards)` is submitted, the Active Directory Query Processor expands the simple filter into a more complex OR wildcard filter that contains all attributes marked as part of the ANR set. The specified filter on a Windows Server 2012 Active Directory with Exchange Server 2010 installed would expand that simple query to:

```
(
(|
  (displayName=joe richards*)
  (givenName=joe richards*)
  (legacyExchangeDN=joe richards)
  (msDS-AdditionalSamAccountName=joe richards*)
  (msDS-PhoneticCompanyName=joe richards*)
  (msDS-PhoneticDepartment=joe richards*)
  (msDS-PhoneticDisplayName=joe richards*)
  (msDS-PhoneticFirstName=joe richards*)
  (msDS-PhoneticLastName=joe richards*)
  (physicalDeliveryOfficeName=joe richards*)
  (proxyAddresses=joe richards*)
  (name=joe richards*)
  (sAMAccountName=joe richards*)
  (sn=joe richards*)
  (&
    (givenName=joe*)
    (sn=richards*)
  )
  (&
    (givenName=richards*)
    (sn=joe*)
  )
)
)
```

As you can see, a very simple query can quickly be expanded into a very large query. For this reason, you should avoid adding additional ANR attributes.

Preserving attributes in a tombstone

The attributes that are retained when an object is deleted (a.k.a. *tombstoned*) are configured through a combination of the `searchFlags` setting and some hardcoded internal functionality. The *preserve on tombstone* `searchFlags` setting is configurable by administrators, so they can choose to add more attributes to what is kept on a tombstoned object. The more attributes you allow the directory to retain on the tombstoned object, the fewer attributes you have to recover through other means after the object is reanimated. If you have the Active Directory Recycle Bin enabled in your forest; however,

there is no value in preserving additional values on a tombstone since tombstones cannot be reanimated once the Recycle Bin is enabled.

Unfortunately, not all attributes can successfully be added to the tombstone when the proper `searchFlags` bit is set. The most obvious examples are linked attributes such as group membership. Linked attributes are handled differently by the directory, and thus there is no way to force them to be retained. If you configure a linked attribute to be preserved, Active Directory will simply ignore the setting.

For more information on tombstones and the Active Directory Recycle Bin, refer to Chapter 18.

The subtree index

Creating this index allows any virtual list view (VLV) operations using a particular attribute as the sort key to be handled in a significantly more efficient manner. It can also prevent a common failure that causes a VLV query to terminate with a "Critical extension unavailable" error in larger directories. By default, when a VLV query is processed, the directory executes the query and stores the result set in a special internal table called the *temp table* so it can be sorted. This table can vary in size, but it can be no larger than the value specified in the `MaxTempTableSize` setting of the Default Query Policy. The value for that setting, by default, is only 10,000 records. If the query result set exceeds the size of the current temp table, you will get the "Critical extension unavailable" error instead of the results. Enabling the subtree container index allows AD to avoid using the temp table, which significantly increases performance and avoids the issue of exceeding the temp table size.

The tuple index

When you create an index, it is optimized for direct lookups and, if the attribute syntax supports it, trailing wildcards—e.g., `(name=joe*)`. If you use medial queries—that is, queries with wildcards anywhere but the end of the string, such as `(name=*oe)`—performance tends to be rather less than optimal. Generally, this is okay, but in the cases where an important application is being significantly impacted due to poor medial query performance, you may want to consider enabling a tuple index for the attribute. This is just like enabling a normal index; you simply enable another bit on the attribute's `searchFlags` mask to specify that a tuple index should be created.

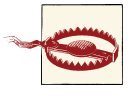
A tuple index is considered an expensive index, and it will increase the Active Directory database (*ntds.dit*) size more than a "normal" index. In addition, new attribute insertion performance will be impacted slightly. This performance hit will not be noticeable for single attribute insertions, but if you are updating a large number of attributes at once, the performance hit may be more noticeable.

Confidentiality

A new bit for the `searchFlags` attribute was defined for Windows Server 2003 Service Pack 1: the *confidential attribute* flag. Any attribute that has this flag enabled requires two permissions in order to be viewed by a trustee (*trustees* are the security principals who are granted permissions): the trustee needs Read Property and Control Access permissions for the attribute. This functionality was put into place primarily to protect sensitive user attributes such as Social Security numbers and other personal information. By default, only the administrators and account operators have full control over all user objects, which means they will be able to view any confidential attributes. Anyone else who has full control over a user object will also be able to view the confidential data, so this is yet another reason to not grant unnecessary rights in the directory. If you have domain controllers in the domain (or global catalogs in the forest, if you are dealing with an attribute in the partial attribute set) that are not running Windows Server 2003 Service Pack 1 or newer, any attributes marked as confidential will still be viewable without the special access rights on those domain controllers or global catalogs.

The confidential attribute capability was added as a workaround to issues that exist in the current security model in Active Directory. Unfortunately, there are a large number of explicit Read Property grant permissions on objects in Active Directory that are terribly difficult to override. This new flag allows you to step in despite all the default grant permissions and quickly deny access to an attribute.

This new function was welcomed with open arms in the Active Directory community until administrators started to realize that Microsoft purposely limited the scope of the functionality by not allowing you to set Category 1 attributes as confidential. Category 1 attributes are many of the attributes defined in the default AD schema, and that list of attributes contains many of the attributes you probably want to make confidential, such as telephone numbers, addresses, employee IDs, and so on. It seems the intent is simply to give AD administrators a way to better secure custom attributes they have added to the directory with schema extensions. This limitation drastically reduces the usefulness of this capability for companies that stick to the default schema.



Modification of `searchFlags` to enable confidentiality on Category 1 attributes is strictly disallowed. If you try to change `searchFlags` on one of these attributes so that the confidential flag is set, you will get an error message.

The default security descriptors on all AD LDS base schema objects are configured with no explicit access control entries (ACEs). The result is very few explicit Read Property grant permissions on objects when they are instantiated, which means you can more easily secure attributes with inherited deny permissions and will not need to depend on the confidential attribute functionality.

Next, we need to discuss the tools that Microsoft has made available starting with Windows Server 2003 Service Pack 1 to handle managing access to confidential attributes. The answer is easy: *none*. In order to grant a trustee the ability to view a specific confidential attribute on an object, a grant ACE with Control Access permission for the specific attribute needs to be added to the ACL of the object.

The GUI tools available for assigning permissions not only do not have the ability to assign this type of permission, but they can't even display the permission if something else grants it. The command-line tool *dsacls.exe* is only marginally better; it can display the permission, but cannot grant the permission. The best that the GUI and the *dsacls.exe* tool can do is assign either full control to the object or *all* control access rights to the object, but neither of these is optimal if you prefer to give the minimum rights necessary to get the job done.

The version of LDP that comes with the Windows Server 2008 (and newer) RSAT tools is able to modify ACLs properly and is the best way to manage access to confidential attributes. Even if your forest is not running Windows Server 2008 or newer, you can use this version of LDP. For more information on modifying the ACL of an object, see Chapter 16.

Attribute change auditing

Starting with Windows Server 2008, the auditing infrastructure has been substantially updated compared to its predecessors. In Windows 2000 and Windows Server 2003, there was a single domain-wide directory service auditing setting called "audit directory service access." When this setting was enabled, all directory services auditing events were enabled. In a busy environment, the consequence of this was a substantial amount of security audit traffic in the event logs, to the point that it could easily become unmanageable and thus impractical to have enabled.

Windows Server 2008 and newer domain controllers separate directory services auditing into four subcategories:

- Directory Service Access
- Directory Service Changes
- Directory Service Replication
- Detailed Directory Service Replication

Of particular interest to us right now is the Directory Service Changes subcategory. We will discuss the new auditing infrastructure in much more detail in Chapter 16.

By default, all attribute changes will continue to be audited as required by the *system ACL* (SACL). In order to control noise, however, you can set bit 9 on an attribute's `searchFlags` mask to disable all change audits for that attribute. The Windows Server

2008 GUI tools do not expose an interactive method to easily set this bit. For directions on setting this bit, reference the sidebar "Controlling Attribute Change Auditing" next.

Controlling Attribute Change Auditing

The GUI tools do not expose an easy mechanism for controlling bit 9 in an attribute's `searchFlags` mask without performing the binary arithmetic by hand. Fortunately, this change can easily be enacted by combining the *adfind* and *admod* tools.

In this example we will disable change auditing for the `description` attribute:

```
adfind -sc s:description searchFlags -adcsv | admod -safety 1 -expand
searchFlags::{{searchFlags:SET:0x0100}}
```

And in this example we will enable change auditing for the `description` attribute:

```
adfind -sc s:description searchFlags -adcsv | admod -safety 1 -expand
searchFlags::{{searchFlags:CLR:0x0100}}
```

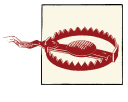
Note that you can apply this pattern to toggle any bit mask flag in Active Directory.

The filtered attribute set

The filtered attribute set is part of the new read-only domain controller (RODC) functionality in Windows Server 2008 Active Directory. RODCs can be configured to not replicate certain attributes in the Active Directory schema. There is an in-depth discussion of RODCs in Chapter 9; however, we will discuss RODCs briefly in the context of the filtered attribute set here.

RODCs were designed with the mentality that the server on which they are running is compromised by default. Consequentially, there are some attributes that we might not wish to have stored on an RODC, as they could contain sensitive information. Examples might be schema extensions that contain application-specific secrets, confidential data such as Social Security numbers, and so forth.

You can apply the process illustrated in the sidebar "Controlling Attribute Change Auditing" on page 93 to control whether or not attributes are included in the filtered attribute set. Instead of toggling bit `0x100`, you should toggle bit `0x200`.



You cannot include attributes that are defined as critical in the `schemaFlagsEx` attribute in the filtered attribute set.

Property Sets and attributeSecurityGUID

Property sets are described in our Chapter 16 discussion on Active Directory security. We mention them here because the creation, modification, and identification of property sets involve the schema partition. Part of the information for a property set is maintained in the Configuration container in the `cn=extended-rights` subcontainer, and the rest is maintained in the schema.

The property sets are defined in the `cn=extended-rights` subcontainer as `controlAccessRight` objects. Two of the attributes of the `controlAccessRight` object link it to the schema. The first is the `appliesTo` attribute; the second is the `rightsGuid`. The `appliesTo` attribute is the string representation of the `schemaIDGUID` attribute of the `classSchema` objects that the property set applies to. The `rightsGuid` is the string representation of the binary GUID stamped on the `attributeSecurityGUID` attribute of `attributeSchema` objects to identify them as members of the property set.

Linked Attributes

Microsoft allows distinguished name attributes with `attributeSyntax` values of `2.5.5.1`, `2.5.5.7`, and `2.5.5.14` to be linked to attributes with an `attributeSyntax` of `2.5.5.1`. These are called *linked attributes* and consist of a forward link and a back link. An example of a pair of linked attributes is `member` and `memberOf`.

Attributes are linked by setting the `linkID` attributes of two `attributeSchema` objects to valid link values. The values must be unique for all `attributeSchema` objects. The value of the forward link is a positive even value, and the value of the back link is the forward `linkID` value plus one, to make it a positive odd value. Attributes must be linked when they are first defined in the schema.

You can use any random `linkID` values as long as they result in a unique `linkID` pair; however, it is *highly* recommended that you autogenerate link IDs. You cannot use autogenerated link IDs in the case that you need your schema extension to support Windows 2000. If you need to support Windows 2000, you should email *schemreg@microsoft.com* to request unique link IDs.

In order to autogenerate link ID pairs, there are four steps you must follow:

1. Create the forward link attribute and populate the `linkID` attribute with the value `1.2.840.113556.1.2.50`.
2. Reload the schema cache.
3. Create the back link attribute and populate the `linkID` attribute with the `lDAPDisplayName` of the forward link attribute you created in step 1.
4. Reload the schema cache.

MAPI IDs

Any attribute that is displayed in the Exchange global address list (GAL) must have a MAPI ID defined in Active Directory. Once you define a MAPI ID for an attribute (by storing it in the `mAPIId` attribute of the `attributeSchema` object), an Exchange administrator can use the Details Template Editor (shown in Figure 5-5) to add the attribute to the address book. The downfall of MAPI IDs is that there is no central authority for issuing MAPI IDs, so, it is possible that there could be a conflict. Fortunately, if your schema master is running Windows Server 2008 or newer, AD can generate a MAPI ID automatically when creating a new attribute. To take advantage of the automatic MAPI ID generation functionality, follow these steps:

1. Create a new attribute and populate the `mAPIId` attribute with the value `1.2.840.113556.1.2.49`.
2. Reload the schema cache.

AD will assign the next available MAPI ID between `61,440` and `65,520`.

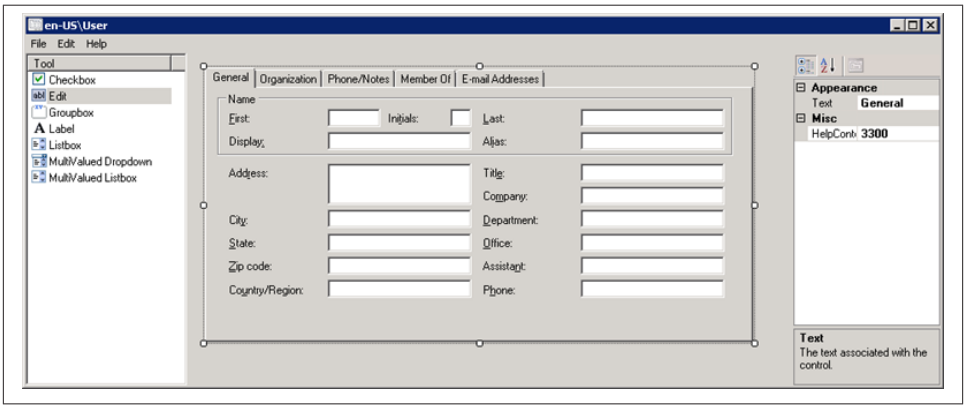


Figure 5-5. The details Template Editor

If you try to create a new attribute and explicitly specify a MAPI ID in the reserved range of `61,440` to `65,520`, AD will return a "Will not perform" error.

Classes (classSchema Objects)

Schema classes are defined as instances of the `classSchema` class. A complete listing of the attributes on the `classSchema` class is available at this link.

Object Class Category and Inheritance

Classes are special in that they can inherit from one another. For example, let's say that we wanted to store two new types of objects in the schema, representing a marketing user and a finance user, respectively. These users both need all the attributes of the existing `user` class as a base. However, the finance user needs three special attributes, while the marketing user needs seven. The extra attributes required by both users do not match in any way. In this example, we can create a `marketing-user` class, a `finance-user` class, and 10 distinctly new attributes. However, rather than having to specify that the `marketing-user` and `finance-user` classes have each of the attributes of the original `user` class individually, all we need to do is specify that the new classes inherit from the `user` class by setting the `subClassOf` attribute of each to `user`. When we do this, both of the new classes inherit every single attribute that the `user` class has. We can then add the extra attributes to each class, and we have two new classes. This example is outlined in Figure 5-6.

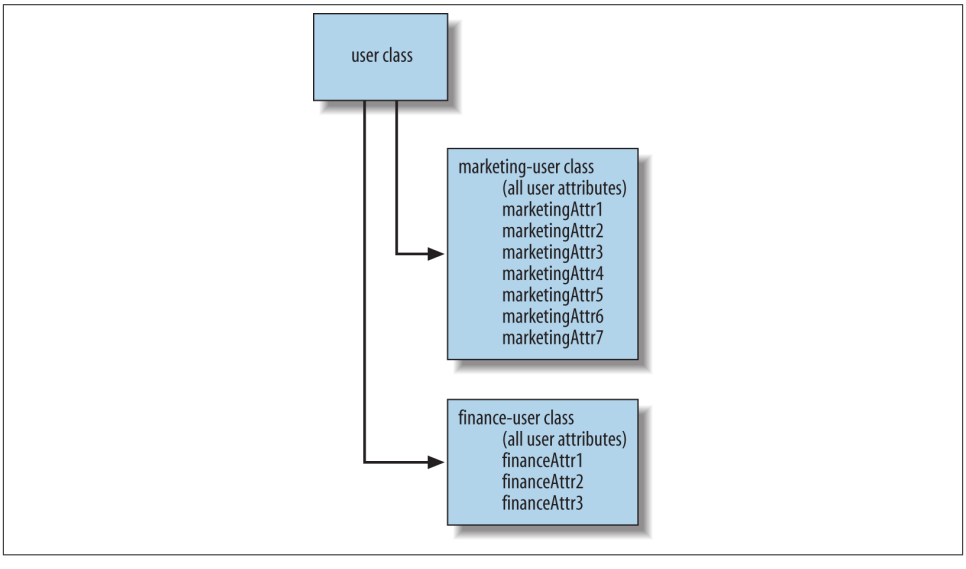


Figure 5-6. Marketing and finance subclasses

You can think of the Active Directory schema as a treelike structure, with multiple classes branching down or inheriting from one base class at the top that has the attributes all objects need to begin with. This class, unsurprisingly enough called `top`, was originally defined in the X.500 specification. Some classes inherit directly from `top`, while others exist much lower down the tree. While each class may have only one parent in this layout, each class may also inherit attributes from other classes. This is possible because

you can create three categories of `classSchema` object, also known as the `objectClassCategory`:

Structural
: If a class is structural, you can directly create objects of its type in Active Directory. The `user` and `group` classes are examples of structural classes.

Abstract
: It is possible that you will want to create a class that inherits from other classes and has certain attributes, but that is not one you will ever need to create instances of directly. This type of class is known as *abstract*. Abstract classes can inherit from other classes and can have attributes defined on them directly. The only difference between abstract and structural classes is that an object that is an instance of an abstract class cannot be created in Active Directory. If you are familiar with an object-oriented programming language, abstract schema classes in Active Directory are analogous to abstract classes in the programming language.

Auxiliary
: An auxiliary class is used to store sets of attributes that other classes can inherit. Auxiliary classes are a way for structural and abstract classes to inherit collections of attributes that do not have to be defined directly within the classes themselves. Auxiliary classes are primarily a grouping mechanism.

The X.500 specifications indicate that an auxiliary class cannot inherit from a structural class, and an abstract class can inherit only from another abstract class.

To comply with the X.500 standards, there are actually four types of `objectClassCategory`. While objects are required to be classified as one of structural, abstract, or auxiliary by the 1993 X.500 specifications, objects defined before 1993 using the 1988 specifications are not required to comply with these categories. Such objects have no corresponding 1993 category and so are defined in the schema as having a special category known as the *88-Class*.

Let's take a look at the `user` and `computer` classes that are used to create user and computer accounts within Active Directory. The `computer` class and `user` class are each structural, which means that you can instantiate objects of those classes directly in Active Directory. The `computer` class inherits from the `user` class, so the `computer` class is a special type of user in a way. The `user` class inherits from the `organizationalPerson` Type-88 class. This means that the total attributes available to objects of class `computer` include not only the attributes defined specifically on the `computer` and `user` classes themselves, but also all the attributes that are inherited from the `organizationalPerson` class. The `organizationalPerson` class is a subclass of the `person` abstract class,

which is a subclass of the abstract `top` class. Recall that there are no classes above `top`; it is the root class. This relationship is outlined in Figure 5-7.

The `user` class that Microsoft needed to define in Active Directory had to be more than just the sum of the X.500 standard parts. After all, Microsoft uses security identifiers (SIDs) to identify users, and these were not contained in the original X.500 standards. So, to extend the attributes that make up a user, Microsoft defined some auxiliary classes and included these in the `user` class makeup. The auxiliary classes are `mailRecipient`, `securityPrincipal`, `posixAccount`, and `shadowAccount`. `mailRecipient` is a collection of attributes that allow a user object to hold information relating to the email address and mail account associated with that user. Similarly, the `securityPrincipal` class holds the SID and other user-related security attributes that Microsoft needed. The `shadowAccount` and `posixAccount` auxiliary classes were added in Windows Server 2003 R2 to support Microsoft Services for UNIX (SFU).

If you were to use a tool such as ADSI Edit, you could see the inheritance and class relationships quite clearly. For example, looking at the `objectClass` attribute of any user object, you would see that at a minimum the values held in this attribute were `top`, `person`, `organizationalPerson`, and `user`. In other words, this attribute indicates that each user object inherits attributes from all these classes. Similarly, for any computer object, the `objectClass` attribute holds `top`, `person`, `organizationalPerson`, `user`, and `computer`. If you were to look at the `subclassOf` attribute on the `computer` class object itself in the schema, you would see the `user` class. The `user` class has a `subClassOf` attribute that indicates `organizationalPerson`, and so on. You can see this relationship in Figure 5-7.

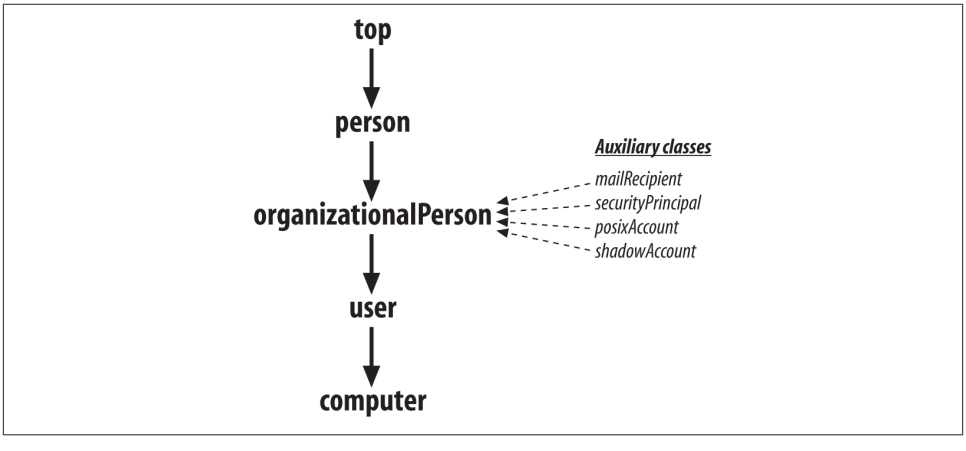


Figure 5-7. Inheritance relationships

Dissecting an Example Active Directory Class

Let's now look at the `user` class in a little more depth. Using a tool like ADSI Edit, we can see the values of each attribute for the user `classSchema` object. Table 5-7 contains a partial listing of the attributes and their values.

Table 5-7. Partial listing of attributes and values for the user class

User attribute's LDAP-Display-Name	User attribute's syntax	Value contained in user's attribute
`adminDescription`	`CASE_IGNORE_STRING`	`User`
`adminDisplayName`	`CASE_IGNORE_STRING`	`User`
`auxiliaryClass`	`CASE_IGNORE_STRING`	`posixAccount shadowAccount`
`cn`	`CASE_IGNORE_STRING`	`User`
`defaultHidingValue`	`BOOLEAN`	`False`
`defaultSecurityDescriptor`	`CASE_IGNORE_STRING`	SDDL text-encoded representation of the default security descriptor
`governsID`	`CASE_IGNORE_STRING`	`1.2.840.113556.1.5.9`
`lDAPDisplayName`	`CASE_IGNORE_STRING`	`user`
`name`	`CASE_IGNORE_STRING`	`User`
`nTSecurityDescriptor`	`SECURITY_DESCRIPTOR`	Binary representation of the security descriptor for the class
`objectCategory`	`DN_STRING`	`cn=Class-Schema,cn=Schema,cn=Configuration, dc=mycorp,dc=com`
`objectClass`	`CASE_IGNORE_STRING`	`top; classSchema` (two values of a multivalued attribute)
`objectClassCategory`	`INTEGER`	`1`
`rDNAttID`	`CASE_IGNORE_STRING`	`cn`
`schemaIDGUID`	`OCTET_STRING`	GUID that uniquely identifies this class
`showInAdvancedViewOnly`	`BOOLEAN`	`True`
`subClassOf`	`CASE_IGNORE_STRING`	`organizationalPerson`
`systemAuxiliaryClass`	`CASE_IGNORE_STRING`	`securityPrincipal mailRecipient`
`systemMayContain`	`CASE_IGNORE_STRING`	Various attributes (see discussion)
`systemPossSuperiors`	`CASE_IGNORE_STRING`	`builtinDomain organizationalUnit domainDNS`

You can learn the following about the `user` class by inspecting these attribute values:

- The name of the class is `User` (`adminDescription`, `adminDisplayName`, `cn`, `name`).
- It is an instance of the `classSchema` class (`objectCategory` and `objectClass`).
- It inherits attributes from both `top` and `classSchema` (`objectClass`).

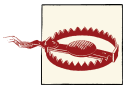
- This object class has a security descriptor governing who can access and manipulate it (`nTSecurityDescriptor`).
- A default security descriptor should be applied to new instances of the `user` class if one is not specified on creation (`defaultSecurityDescriptor`).
- The instances of the `user` class are visible in normal browsing (`defaultHidingValue`).
- The `user` class itself is not hidden from casual browsing (`showInAdvancedViewOnly`).
- The `user` class has an OID of 1.2.840.113556.1.5.9 (`governsID`).
- It inherits attributes not only from `top` and `classSchema` but also from `securityPrincipal` and `mailRecipient` (`objectClass` and `systemAuxiliaryClass`).
- The `shadowAccount` and `posixAccount` auxiliary classes can be associated with user objects.
- The `user` class is a direct subclass of the `organizationalPerson` class (`subClassOf`).
- When connecting to instances of the class via LDAP, the two-letter prefix used should be `cn` (`rDNAttID`).
- This class can be created directly under only three different parents in Active Directory (`systemPossSuperiors`).
- The class is structural (`objectClassCategory`).
- There are a large number of attributes that instances of the `user` class can have values for (`systemMayContain`).

How inheritance affects mustContain, mayContain, possSuperiors, and auxiliaryClass

Let's look at `mustContain`, `mayContain`, `auxiliaryClass`, `possSuperiors`, and their system attribute pairs. You can see that the only values set are `systemPossSuperiors`, `systemMayContain`, and `systemAuxiliaryClass`. These were the values set on the initial creation of the `user` class and they cannot be changed. You can tell that there were no mandatory attributes set at the creation of the original class because the attribute `systemMustContain` is not listed. If you later wished to add an extra set of attributes or a new optional attribute to the `user` class, you could use `auxiliaryClass` or `mayContain` and modify the base definition. This might occur if, for example, you install the Exchange schema updates in your forest. If you were to do this, the `user` class would be directly modified to include three of these Exchange-related auxiliary classes in the `auxiliaryClass` attribute:

- msExchMailStorage
- msExchCustomAttributes
- msExchCertificateInformation

The attributes that are required when you create a new user are not listed in the mustContain attribute. That's because objectSID, sAMAccountName, and the other attributes are inherited from other classes that make up this one. The mustContain attributes can be defined directly in auxiliaryClass or systemAuxiliaryClass, or they can be defined on the classes from which it inherits further up the tree. Both sAMAccountName and objectSID, for example, are defined on the securityPrincipal class.



Do not mistake attributes that a class must contain with the attributes that you must explicitly set on object instantiation. Unfortunately, there is no effective way to programmatically determine what attributes you need to set on an object when you create an instance of the class. Some of the attributes that an object must contain are system-owned and can only be set by the system; other attributes are optional and will be populated automatically; and finally, some attributes are actually required to be specified by the object's creator. To confuse the situation even more, various versions of the OS or AD LDS change the requirements.

The same principle applies to the mayContain attribute. The entire set of attributes that the class may contain is available only when you recurse back up the tree and identify all the inherited mayContain attributes on all inherited classes.

The possSuperiors attribute, on the other hand, can be made up of only those items defined directly on the class, those defined on the class identified in the subClassOf attribute, or those defined on any other classes identified in the subClassOf attributes continuing up the subClassOf tree. If that was too confusing, try this: an instance of the user class can have possSuperiors from itself, from the organizationalPerson class defined in the subClassOf attribute, from the person class (identified in the organizationalPerson class's subClassOf attribute), and from top (identified in the person class's subClassOf attribute).

Viewing the user class with the Active Directory Schema snap-in

Take a look at Figure 5-8. This shows the user class viewed with the Active Directory Schema snap-in. You can see the relevant general user data.

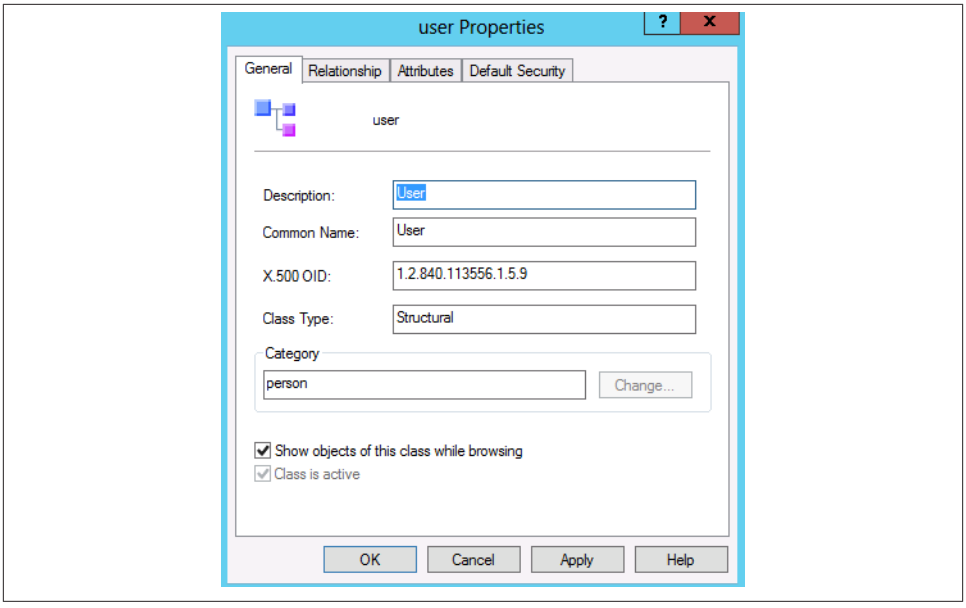


Figure 5-8. User class schema entry general settings

Notice that quite a bit of the `user` class is not configurable after the initial configuration, including:

- `governsID`
- `subClassOf`
- `schemaIDGUID`
- `systemMustContain`
- `rDNAttID`
- `systemPossSuperiors`
- `objectClassCategory`
- `systemMayContain`
- `systemOnly`
- `systemAuxiliaryClass`
- `objectClass`
- `subClassOf`

To see the so-called relationship settings (`subClassOf`, `auxiliaryClass`, `possSuperiors`, `systemAuxiliaryClass`, `systemPossSuperiors`), look at Figure 5-9. In this screen,

you can see that the `user` class in this schema is inheriting attributes from the five auxiliary classes.

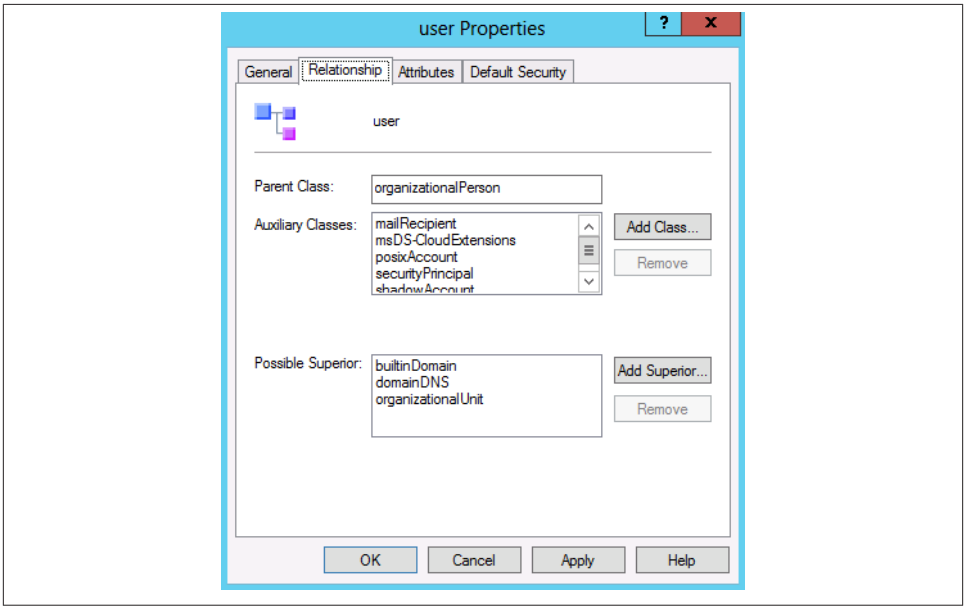


Figure 5-9. User class schema entry relationship settings

The third and final screen of interest to us here is the Attributes tab for the `user` class, displayed in Figure 5-10. This shows the `mustContain`, `systemMustContain`, `mayContain`, and `systemMayContain` attributes of the `user` class.

You can also view the attributes attached to a class by browsing to the class in the tree on the lefthand side of the Active Directory Schema MMC snap-in. All of the attributes of the class will be listed in the main pane to the right.

Dynamically Linked Auxiliary Classes

Dynamically assigning auxiliary classes to individual objects instead of to an entire class of objects in the schema is a feature common in many LDAP directories. Having the dynamic auxiliary class mechanism provides much more flexibility for application developers who may want to utilize existing structural and auxiliary classes, but do not want to extend the schema to define such relationships.

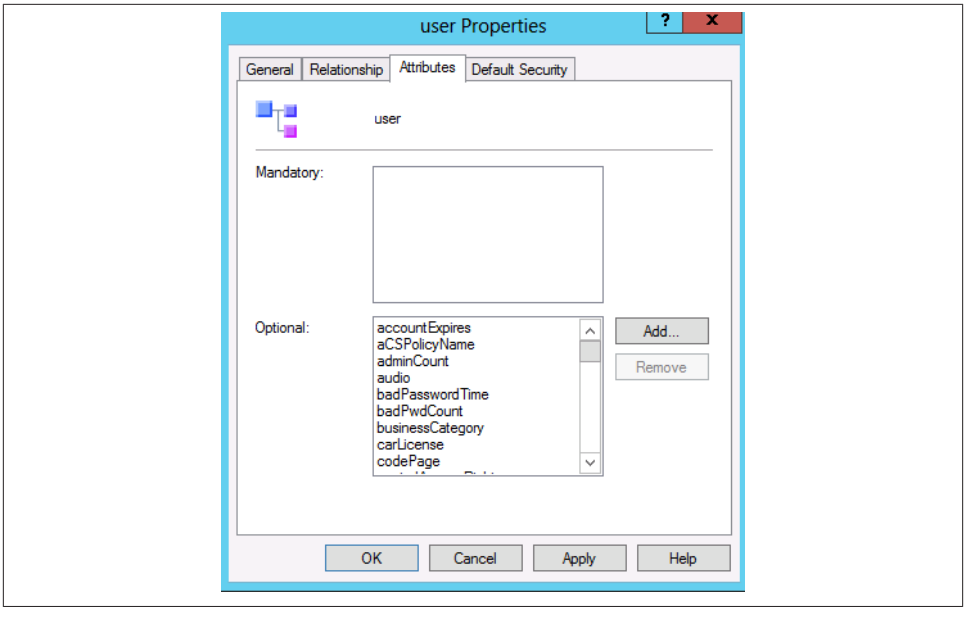


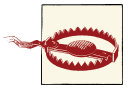
Figure 5-10. User class schema entry attribute settings

Dynamic linking of auxiliary classes requires your forest to be running at the Windows Server 2003 forest functional level, or higher.

To dynamically link an auxiliary class to an object, you only need to modify the `objectClass` attribute of the object to include the name of the auxiliary class. Any auxiliary class can be used, provided that all `mustContain` and `systemMustContain` attributes contained within the auxiliary class are set at the same time. You can also remove a dynamically linked auxiliary class by clearing any values that have been set for attributes defined by the auxiliary class and then removing the auxiliary class name from the object's `objectClass` attribute.

Now let's illustrate why dynamically linking auxiliary classes is a good idea. Assume we have a forest with several domains, each representing divisions within a company. Each division manages its own user objects. One of the divisions, named Toasters, wants to assign additional attributes to its user objects. These new attributes would only apply to employees within the Toasters division. Under Windows 2000, the only way to accomplish this would be to create the new attributes in the schema, create a new auxiliary class, and include the new attributes in the auxiliary class. At that point, the new auxiliary class could be added to the `auxiliaryClass` attribute of the user `classSchema` object. Every user object contained within the forest would then have the new attributes

available. If each division wanted to do something similar, you can see how the number of attributes on all user objects within the forest could grow very quickly and unnecessarily. With Windows Server 2003, you would still create the new attributes and auxiliary classes in the schema, but you would not modify the `auxiliaryClass` attribute of the user object. Instead, each division would dynamically link its auxiliary class to *its* user objects. This provides for a cleaner and much more efficient implementation than was possible under Windows 2000.



When you dynamically link an auxiliary class to an object, the auxiliary class is listed in the `objectClass` attribute for the object. When an auxiliary class is statically linked, the auxiliary class is not listed in the `objectClass` attribute. This can cause issues with applications that need to determine available attributes on an object and only look at the schema definitions of an object class or at the `objectClass` attribute of the object itself.

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, as well as the various elements that must be unique in an Active Directory schema extension (such as prefix names and link IDs).

We then detailed how an attribute and its syntax are 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 starting in Windows Server 2003 and why this is significant.

Chapter 17 builds on what you've learned here to demonstrate how you can design and implement schema extensions.

CHAPTER 6

Site Topology and Active Directory Replication

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

This chapter looks at the basics of how sites and replication work in Active Directory. In Chapter 14, 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 provide details on how to effectively design and tailor sites, site links, and replication in Active Directory.

Site Topology

The Active Directory *site topology* is the map that describes the network connectivity, Active Directory replication guidelines, and locations for resources as they relate to the Active Directory forest. The major components of this topology are sites, subnets, site links, site link bridges, and connection objects. These are all Active Directory objects that are maintained in the forest's Configuration container; this allows the information to be locally available on all domain controllers so the DCs can communicate properly.

Site and Replication Management Tools

Microsoft provides the Active Directory Sites and Services MMC snap-in (*dssite.msc*) to help you manage the replication topology. This snap-in allows you to drill down into the Sites container, which holds all the site topology objects and connection objects. The Sites container is physically located directly under the Configuration container in the Configuration naming context. With the Sites and Services snap-in, you can create new sites, subnets, links, bridges, and connections, as well as setting replication schedules, metrics for each link, and so on.

In addition to the Sites and Services snap-in, the *repadmin* command-line tool is an incredibly useful tool and one you should be familiar with. Windows Server 2012 adds a number of PowerShell cmdlets for managing replication as well. To see all of the replication cmdlets, run *Get-Command *-adrep** from Windows PowerShell.

If you are looking for *repadmin* on Windows 2000 or Windows Server 2003, you'll find it in the Windows Support Tools download available from Microsoft.

Subnets

A *subnet* is a portion of the IP space of a network. Subnets are described by their IP network addresses combined with a subnet mask measured in bits. For instance, the subnet mask 255.255.255.0 is a 24-bit subnet mask. If you had a 24-bit mask for the 10.5.20.0 network, your subnet object would be described as 10.5.20.0/24. The subnet objects in Active Directory are a logical representation of the subnets in your environment; they may, but do not necessarily have to, reflect your actual physical subnet definitions. For example, you may have a building that has two 24-bit subnets of 10.5.20.0 and 10.5.21.0, which for the purposes of Active Directory could be included in a single AD subnet of 10.5.20.0/23, which specifies a 23-bit subnet mask.

These examples all represent IP version 4 (IPv4) subnets. Windows Server 2008 introduced support for IP version 6 (IPv6) subnets and domain controllers in Active Directory. The same concepts for subnetting apply; however, the addresses and subnets are much larger. IPv4 uses 32-bit addressing whereas IPv6 uses 128-bit addressing.

You must define subnet information in the directory, because the only key available for determining relative locations on a network is the IP addresses of the machines. The subnets are, in turn, associated with sites. Without these definitions, there is no way for a machine to make an efficient determination of what distributed resources it should

try to use. By default, no subnets are defined in Active Directory. For more information on defining subnets in Active Directory, see the sidebar "Adding Subnets to a Site", next.

Adding Subnets to a Site

When adding subnets to sites via the Active Directory Sites and Services snap-in, you must enter the name of the subnet in the form *network address/bits masked*; e.g., 10.186.149.0/24 is network address 10.186.149.0 with subnet mask 255.255.255.0. Beginning with Windows Server 2008, the dialog supports entry of both IPv4 and IPv6 subnets.

The "bits masked" component of the subnet name is the number of bits set in the subnet mask for that subnet. It can be between 0 and 32. The subnet mask is made up of four *octets* or bytes (four sets of eight bits). To convert the subnet mask to bits, convert each octet from the subnet mask to binary. The subnet mask 255.255.255.0 is 11111111.11111111.11111111.00000000 in binary, which uses 8+8+8 bits (i.e., 24) to define the subnet mask. A subnet mask of 255.255.248.0 would be 11111111.11111111.11111000.00000000, which is 8+8+5, or 21.

If subnets and IP addresses mean very little to you, check out Chuck Semeria's whitepaper "Understanding IP Addressing: Everything You Ever Wanted to Know" at *http://holdenweb.com/static/docs/3comip.pdf*. The whitepaper covers both IPv4 and IPv6.

It is very important to make sure that you have up-to-date subnet information stored in Active Directory any time you are working with a multisite environment. If subnet information is not accurate, unnecessary WAN traffic may be generated as domain members authenticate to domain controllers in other sites. Domain controllers will also log events warning you of missing subnet definitions.

One strategy for addressing missing subnet definitions is the use of *supernets*. Supernet is a term used to describe a single subnet that encompasses one or more smaller subnets. For example, you may define a subnet object of 10.0.0.0/8, as well as many subnets in the same space with larger subnet masks, such as 10.1.0.0/16, 10.1.2.0/24, or even 10.1.2.209/32. When Active Directory works out which subnet a machine is associated with, it chooses the most specific subnet defined. So, if the IP address were 10.1.2.209, the subnet 10.1.2.209/32 would be used; but if the IP address were 10.2.56.3, the 10.0.0.0/8 subnet would be used.

You may be wondering why someone would do this. The supernet objects can be assigned to any site just like any other subnet object. This means you can assign these "catchall" subnets to hub-based sites for machines that are running from subnets that are not otherwise defined. This causes the machines to use the hub resources rather than randomly finding and selecting resources to use.

Figure 6-1 shows a perfect example of an organization whose subnet design lends itself well to the deployment of supernets in Active Directory. There are three hub sites in this topology—Chicago, São Paulo, and London. If you look closely, all of the subnets under Chicago can be summarized as 10.1.0.0/16, all of the subnets under London can be summarized as 10.2.0.0/16, and all of the subnets under São Paulo can be summarized as 10.3.0.0/16. By creating each of these subnets in Active Directory and associating them with their respective hub sites, you will create a safety net for new subnets that might come up at each of the locations.

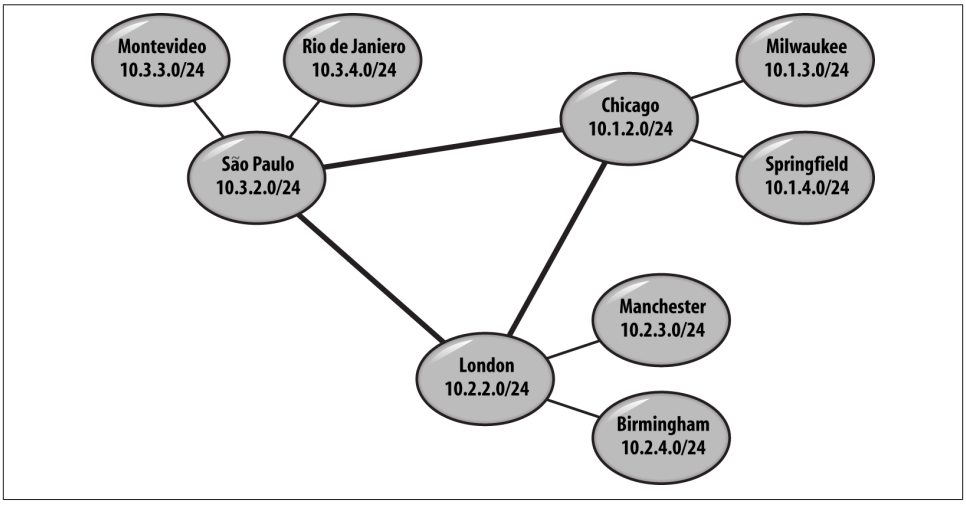


Figure 6-1. Sample site and subnet topology

Managing subnets

You can create subnets with Active Directory Sites and Services or with Windows PowerShell. To create a subnet with Active Directory Sites and Services, browse to the Subnets container under Sites. Right-click and click New Subnet. You can enter an IPv4 or IPv6 subnet address in the Prefix text box, as shown in Figure 6-2. You must also associate the subnet with a site. In Figure 6-2, we are associating the subnet 172.16.1.0/24 with the Chicago site.

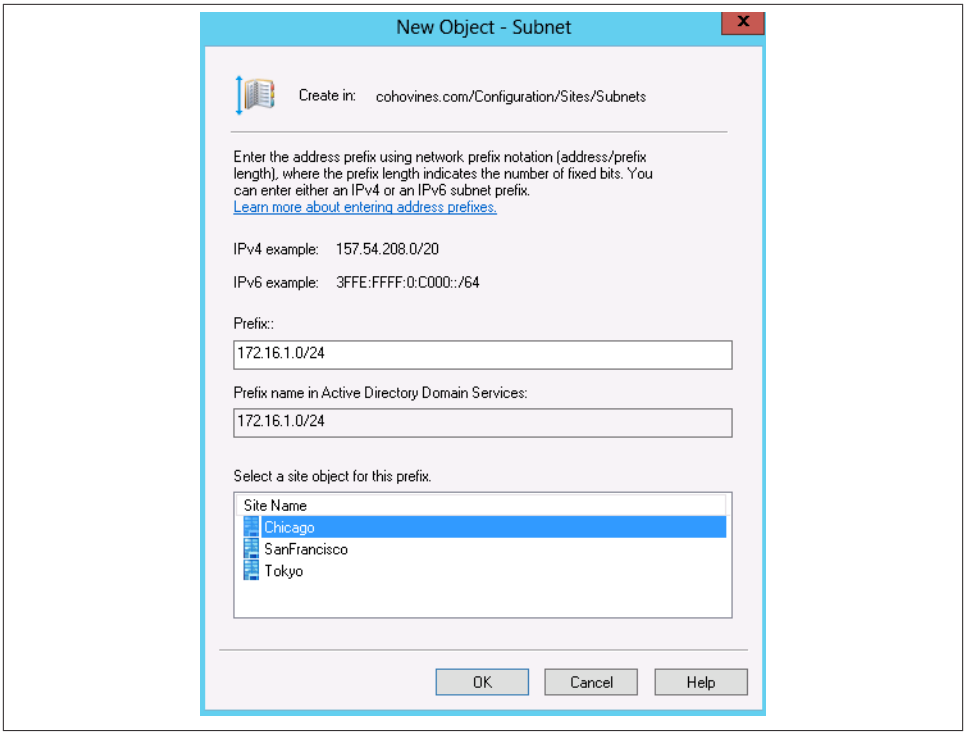


Figure 6-2. Creating a subnet with Active Directory Sites and Services

To create the same subnet with Windows PowerShell, you will need to be using the Windows Server 2012 (or Windows 8) Remote Server Administration Tools (RSAT) or running the command directly on a Windows Server 2012 server. The command to use is:

```
New-ADReplicationSubnet -Name "172.16.1.0/24" -Site "Chicago"
```

If you need to change the site that a subnet is associated with, you can do that from the properties of the subnet, or with the *Set-ADReplicationSubnet* cmdlet.

Troubleshooting subnet data problems

When your forest doesn't have up-to-date subnet information, there are a few things that happen on your domain controllers. First and foremost, clients start connecting to DCs that are not logically near them on the network. This leads to increased WAN traffic and poor client performance. Next, your DCs will start periodically logging events in the event log similar to the one following:

```
Event ID: 5807
Source: NETLOGON
User: N/A
```

```
Computer: DC01
Description:
During the past %1 hours there have been %2 connections to this Domain Controller
from client machines whose IP addresses don't map to any of the existing sites in
the enterprise. Those clients, therefore, have undefined sites and may connect to
any Domain Controller including those that are in far distant locations from the
clients.

A client's site is determined by the mapping of its subnet to one of the existing
sites. To move the above clients to one of the sites, please consider creating sub-
net object(s) covering the above IP addresses with mapping to one of the existing
sites.

The names and IP addresses of the clients in question have been logged on this
computer in the following log file '%SystemRoot%\debug\Netlogon.log' and, poten-
tially, in the log file '%SystemRoot%\debug etlogon.bak' created if the former log
becomes full. The log(s) may contain additional unrelated debugging information.

To filter out the needed information, please search for lines which contain text
  'NO_CLIENT_SITE:'. The first word after this string is the client name and the
second word is the client IP address.
The maximum size of the log(s) is controlled by the following registry DWORD value
  'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters\
LogFileMaxSize'; the default is %3 bytes. The current maximum size is %4 bytes.

To set a different maximum size, create the above registry value and set the
desired maximum size in bytes.
```

Finally, every client that connects to a DC from an IP address without a corresponding subnet in AD will get logged in the DC's *netlogon.log* file, located at *%windir%\debug\netlogon.log*. Log entries for missing subnets will look like this:

```
11/12 05:45:29 BRIANLAB: NO_CLIENT_SITE: PC06 192.168.4.111
11/12 06:06:12 BRIANLAB: NO_CLIENT_SITE: PC03 192.168.4.113
```

In the preceding example, two machines, PC06 and PC03, have connected from IP addresses (192.168.4.111 and 113) that don't have a corresponding subnet in Active Directory. The format of the log entry is `Date Time Domain: NO_CLIENT_SITE: Machine-Name IP-Address`.

The IP address in the preceding log snippet could be an IPv4 or IPv6 address.

If you have IPv6 deployed and your clients have both IPv4 and IPv6 IPs (known as dual stack), you'll need to make sure that AD contains subnet mappings for both the IPv4 and IPv6 subnets. If you only have one or the other configured for a client, you'll start seeing this event periodically:

```
Event ID: 5810
Source: NETLOGON
User: N/A
Computer: DC01
Description:
During the past %1 hours, this domain controller has received %2 connections
from dual-stack IPv4/IPv6 clients with partial subnet-site mappings. A client
has a partial subnet-site mapping if its IPv4 address is mapped to a site but
its global IPv6 address is not mapped to a site, or vice versa. To ensure correct
behavior for applications running on member computers and servers that rely on
subnet-site mappings, dual-stack IPv4/IPv6 clients must have both IPv4 and global
IPv6 addresses mapped to the same site. If a partially mapped client attempts
to connect to this domain controller using its unmapped IP address, its mapped
address is used for the client's site mapping.

The log files %SystemRoot%\debug\netlogon.log or %SystemRoot%\debug\netlogon.bak
contain the name, unmapped IP address and mapped IP address for each partially
mapped client. The log files may also contain unrelated debugging information.
To locate the information pertaining to partial-subnet mappings, search for
lines that contain the text 'PARTIAL_CLIENT_SITE_MAPPING:'. The first word after
this text is the client name. Following the client name is the client's unmapped
IP address (the IP address that does not have a subnet-site mapping) and the
client's mapped IP address, which was used to return site information.

USER ACTION
Use the Active Directory Sites and Services management console (MMC) snap-in
to add the subnet mapping for the unmapped IP addresses to the same site being
used by the mapped IP addresses. When adding site mappings for IPv6 addresses,
you should use global IPv6 addresses and not for instance temporary, link-local
or site-local IPv6 addresses.

The default maximum size of the log files is %3 bytes. The current maximum
size is %4 bytes. To set a different maximum size, create the following registry
DWORD value to specify the maximum size in bytes:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
  \Netlogon\Parameters\LogFileMaxSize
```

You'll also start seeing entries in your *netlogon.log* file that contain the term `PARTIAL_CLIENT_SITE_MAPPING` instead of `NO_CLIENT_SITE`. This entry will tell you the IPv4 or IPv6 address that matched a subnet in Active Directory, and the opposite address that did not. That is, if a client connected with an IPv6 address but that client's IPv4 address matches a subnet in Active Directory, the log will advise you of the unmapped IPv6 address.

Check out this website for a sample script to help you track down subnet mapping issues.

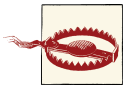
Sites

An Active Directory *site* is generally defined as a collection of well-connected AD subnets. You use sites to group subnets together into logical collections to help define replication flow and resource location boundaries. Active Directory uses sites directly to generate its replication topology, and also to help clients find the "nearest" distributed resources to use in the environment (such as DFS shares or domain controllers). The client's IP address is used to determine which Active Directory subnet the client belongs to, and then that subnet information, in turn, is used to look up the AD site. The site information can then be used to perform DNS queries via the DC locator service to determine the closest domain controller or Global Catalog. For a discussion of the DC locator process, refer to "DC Locator" on page 186.

Most members of a domain dynamically determine their site when they start up, and they continue to validate what site they are in in the background. This allows administrators to make modifications to the site topology and have them take effect properly in relatively short order with the least amount of manual work. Domain controllers, on the other hand, select their site when they are promoted and will not automatically change unless an administrator wants them to become part of another site. Moving a domain controller to another site is an administrative task that is most easily performed via the Active Directory Sites and Services tool.

By default, there is one site defined in Active Directory, the *Default-First-Site-Name* site. If there are no subnet objects defined, all members of the domain are "magically" assumed to be part of this initial site, or any other single defined site if you have replaced the default site with another site. Once there are multiple site objects, or after subnet objects are defined and assigned, the "magic" feature goes away and subnet objects must be defined for the subnets in which domain members reside. There is nothing special about this initial site other than that it is the first one created; you can rename it as you see fit. You can even delete it, as long as you have created at least one other site and moved any domain controllers located within the *Default-First-Site-Name* site to another site.

Multiple sites can be defined for a single physical location. This can allow you to better segregate which resources are used for which requestors. For instance, it is common practice in large companies to build a separate site just to harbor the Microsoft Exchange 2000 and 2003 servers and the global catalogs that are used to respond to Exchange and Outlook queries. This allows an administrator to easily control which GCs are used without having to hardcode preferred GC settings into Exchange. You can define the subnets as small as you need them, including down to a single IP address (32-bit subnet), to place servers in the proper site.



It is no longer a recommended best practice to place Exchange servers in a separate site, due to the changes in Exchange's mail-routing behavior beginning with Exchange 2007. If you are deploying Exchange in your environment you will need to work closely with your Exchange administrators to ensure you have enough Global Catalog capacity in the sites that support Exchange.

Managing sites

Much like subnets, you can create sites with the aptly named Active Directory Sites and Services MMC snap-in, or with Windows PowerShell. When you create a site, as shown in Figure 6-3, you'll be prompted to pick a site link that defines the path linking the site to the rest of the topology. When using the GUI, there is a bit of a Catch-22 because in order to create a site you need to pick a site link, but as you'll see, when you're ready to create a site link, you'll need to pick the sites that are linked. Thus, when you create the site, it is fine to temporarily pick a site link (such as the default shown in Figure 6-3) and then make changes once you have the site links ready.

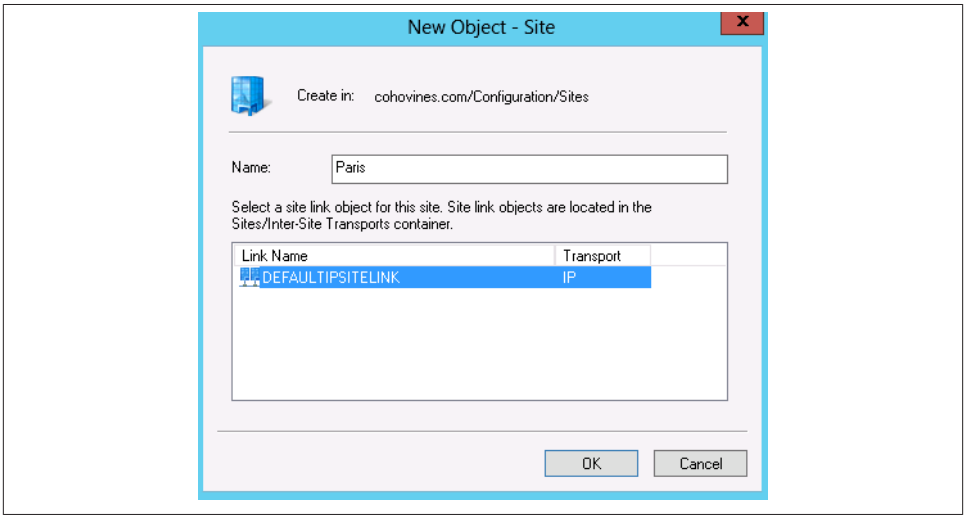


Figure 6-3. Creating a site with AD Sites and Services

To create the same site with Windows PowerShell, you can run the following command:

```
New-ADReplicationSite -Name "Paris"
```

Note that when you use Windows PowerShell, you don't have to specify a site link when you create a site. The circular dependency is a function of the MMC snap-in, not Active Directory.

If you want to see the list of subnets that are associated with a site, they are displayed on the properties of the site in the AD Sites and Services MMC snap-in.

Site Links

Site links allow you to define what sites are connected to each other and the relative cost of the connection. When you create a site link, you specify which sites are connected by this link and what the cost or metric of the connection is in a relative-costing model. For instance, three sites—A, B, and C—could be connected to each other, but because you understand the underlying physical network, you might feel all traffic should be routed through the A site. This would require you to configure to two site links: A to B and A to C, as shown in Figure 6-4.

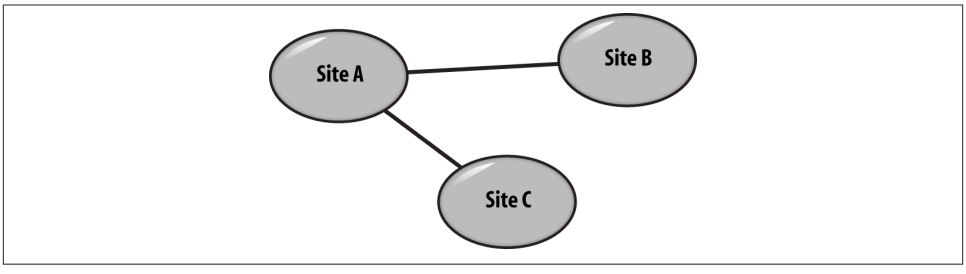


Figure 6-4. Sample site topology

If at a later time additional physical network connections were established between B and C, you could set up one more site link and connect B and C together. If you configure all three site links with an equal cost—say, 100—traffic could then flow across the B-to-C link instead of from B to A to C. This is because the total cost to use the B-to-C link would be 100 and the total cost to use the B-to-A-to-C route would be 200, which is more expensive. This scenario is demonstrated in Figure 6-5.

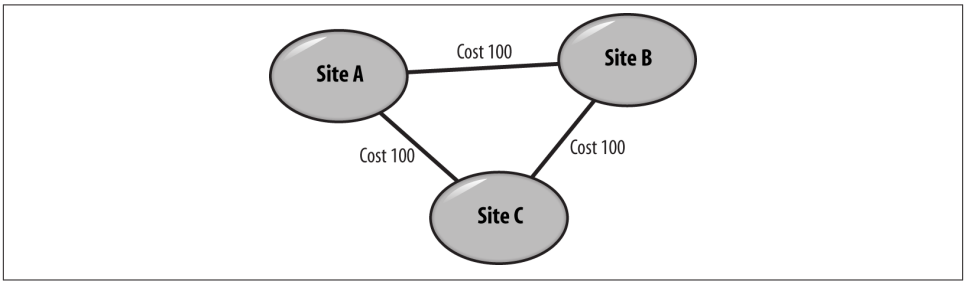


Figure 6-5. Sample site topology with equal costs

If you prefer that traffic to still flow through Site A, your total cost from B to A to C (and the reverse) should be lower than your cost from B to C. So, you would need to set the B-to-C link cost to something greater than 200—like 300, as shown in Figure 6-6. In this scenario, the B-to-C site link would only be used if the connection could not go through Site A for some reason.

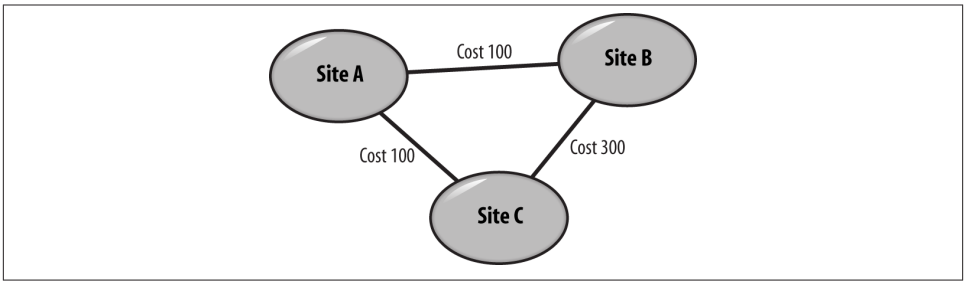


Figure 6-6. Sample site topology with unequal costs

Site links can come in two replication flavors: IP and SMTP. An IP site link actually represents a a remote procedure calls (RPC)-style replication connection. This is by far the most commonly used type of site link; in fact, the default site link created by Active Directory is an IP site link called *DEFAULTIPSITELINK*. The other site link type is an SMTP site link.

SMTP site links are a special SMTP-based (Simple Mail Transfer Protocol—think email) replication protocol. SMTP replication is used to communicate with sites that have very poor or unreliable network connections (such as extremely high latency or unpredictable availability) that cannot support an RPC replication connection. Due to security concerns, however, only certain naming contexts can be replicated across SMTP site links. Only the Configuration NC, Schema NC, and read-only Global Catalog NCs can be replicated via SMTP. This means that you must still replicate the domain NC via the more common IP replication protocol (with RPC).

Besides defining which replication protocols should be used and which sites should be replicating with what other sites, site links control domain controller and Global Catalog coverage of sites that do not have local DCs or GCs. This behavior is called *auto site coverage*.

Consider the diagram in Figure 6-7 showing Sites A, B, and C. Site A has domain controllers for Dom1 and Dom2, but Site B only has a DC for Dom1 and Site C for Dom2. How does a client decide which DC it should use for a domain that doesn't have a domain controller in the local site? Logically, you can look at it and say, "If someone needs to log on to Dom1 while in Site C, the client should talk to Site A." But Active Directory cannot make this intuitive leap; it needs to work with cold hard numbers, and that is what the site link metrics do: give the directory the information needed to make the

calculations to determine which sites a given domain controller should cover in addition to its own site.

Active Directory will automatically register the necessary covering DNS records for sites that do not have domain controllers, based on the site link topology. The relationship between Active Directory and DNS is covered in detail in Chapter 8. For information on disabling auto site coverage, see this website.

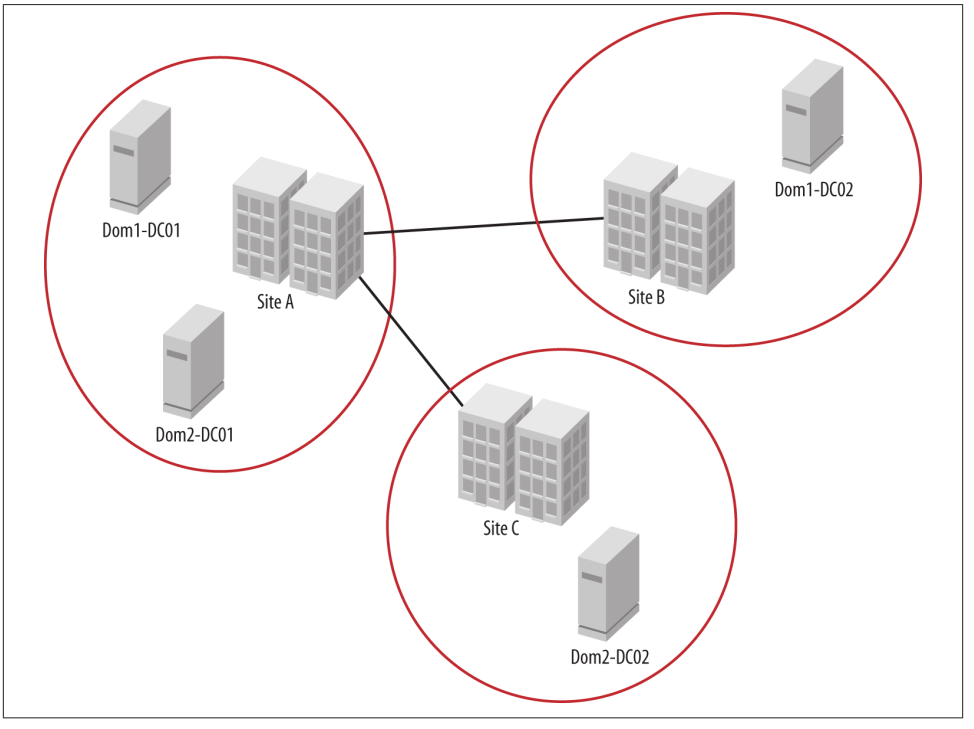


Figure 6-7. Sample site topology requiring automatic site coverage

Managing site links

The place to create site links in the AD Sites and Services MMC snap-in is a bit hard to find: the option is buried under *Sites\Inter-Site Transports\IP*, as shown in Figure 6-8.

If you are using SMTP replication and you need to create an SMTP site link, you will need to instead create the site link from inside the *SMTP* folder pictured in Figure 6-8.

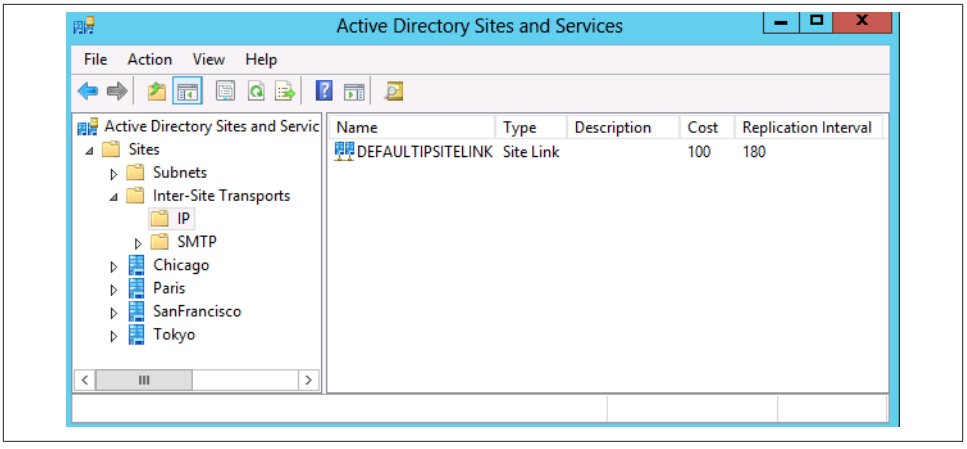


Figure 6-8. Managing IP site links with AD Sites and Services

Once you've found the correct location, you can right-click and click New Site Links.... You'll need to provide a name for the site link and select two or more sites to include in the site link. While AD supports more than two sites in a link, we generally recommend that you stick with two sites per link to make troubleshooting and understanding the expected topology easier.

In Figure 6-9, we create a site link between Chicago and Paris. Once you create the site link, consider adding a description ordered in the opposite order of the name. For example, in Figure 6-9, we created a site link called *Chicago-Paris*. If you set the description to *Paris-Chicago*, you'll be able to sort on either side of a site link by sorting on either the Name or Description the columns (shown in Figure 6-8).

To create the same site link with Windows PowerShell, run this command:

```
New-ADReplicationSiteLink -Name "Chicago-Paris" -Description "Paris-Chicago" `
-SitesIncluded @("Chicago","Paris") -Cost 100 -ReplicationFrequencyInMinutes 15
```

There is one piece of syntax here that may be new to you. In the *SitesIncluded* parameter, we have to provide PowerShell with an *array*. Arrays enable you to pass a list to a particular argument. PowerShell provides a shortcut to creating an array in the form of wrapping the list inside of the *@()* syntax shown in this example.

After you create a site link, chances are you'll want to set some properties of the site link, such as the cost, replication frequency, or possibly the schedule. Figure 6-10 shows the properties of the Chicago-Paris site link. In this scenario, we are setting the cost to 100 and the replication interval to 15 minutes. You can manage the schedule with the Change Schedule button shown in Figure 6-10. Keep in mind that the site link's schedule defines when replication can *begin*. Once replication has begun, it will continue until it has finished.

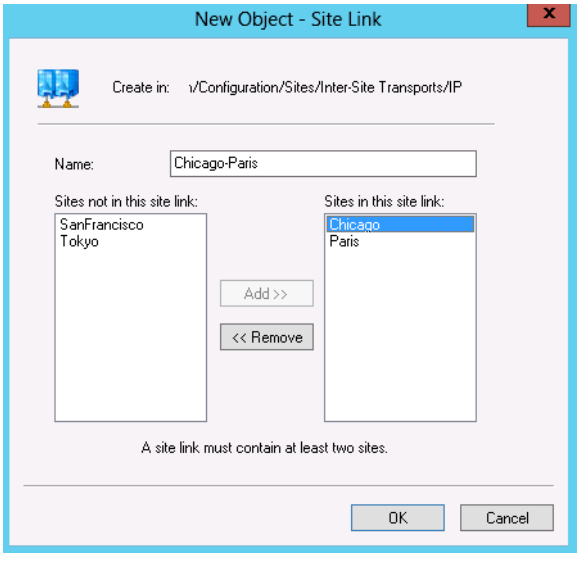


Figure 6-9. Creating a site link between Chicago and Paris

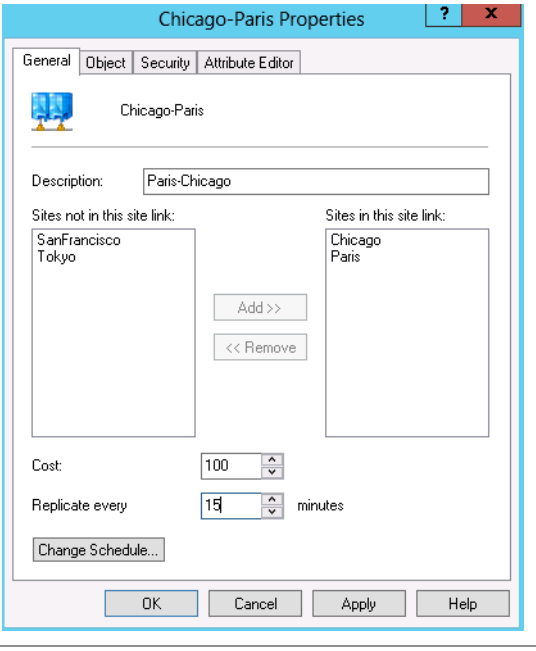


Figure 6-10. Site link properties

Site Link Bridges

By default, Active Directory assumes that the network paths between all of your sites are transitive. In other words, if you reference Figure 6-4, Active Directory assumes that domain controllers in Site C can talk *directly* to domain controllers in Site A. In some networks, this behavior is undesirable since the network does not permit this transitivity. Sometimes in the context of a networking discussion, people will refer to this scenario as the network not being *fully routed*. When you find yourself in this scenario, you may need to disable the "Bridge all site links" option for your forest. This option is accessible by right-clicking the *Sites\Inter-Site Transports\IP* folder in Active Directory Sites and Services, and going to Properties. The same setting is independently available for the SMTP replication transport.

If you have disabled "Bridge all site links," you may need to define site link bridges in your directory. A site link bridge contains a set of site links that can be considered transitive. In other words, any of the sites contained in the list of site links that you are bridging can communicate directly. Referring to Figure 6-4 again, if the site links from A to C and A to B are bridged, we are telling Active Directory that direct communication between a domain controller in Site C and a domain controller in Site B is permissible.

We will discuss site link bridges in more detail in Chapter 14. You can create site link bridges from the same location as site links. Windows PowerShell offers the *New-ADReplicationSiteLinkBridge* cmdlet to script the creation of site link bridges.

Connection Objects

The final piece of the topology puzzle is the *connection object*. A connection object specifies which domain controllers replicate with which other domain controllers, how often, and which naming contexts are involved. Unlike sites, subnets, and site links, which you generally need to manually configure, connection objects are generally managed by the domain controllers themselves. The idea is that you should logically construct the site topology with good definitions for sites, subnets, and site links, and Active Directory will be able to figure out the best way to interconnect the actual domain controllers within and between the sites.

It is occasionally not possible to allow AD to manage all of these connections, but it is a very good goal to work toward, and you should endeavor not to modify or supplement connection objects unless you have no other choice. Earlier versions of Active Directory were not able to properly load balance replication connections between domain controllers, so a domain controller in a hub site could become overwhelmed with replication traffic from spoke domain controllers. This scenario often caused administrators to opt to attempt to manage replication connections manually or with the Active Directory Load Balancing (ADLB) tool. Fortunately, beginning with Windows Server 2008, Active Directory gained the ability to automatically load balance replication connection to

read-only domain controllers (RODCs). Windows Server 2008 R2 extended this capability to load balancing of replication connections with all types of domain controllers.

You can view connection objects with AD Sites and Services as well as the *Get-ADReplicationConnection* cmdlet. Both the MMC snap-in and Windows PowerShell enable you to make changes to connection objects, and AD Sites and Services allows you to create new connection objects. When you manually create or modify a connection object, Active Directory will no longer automatically manage that connection object. With this in mind, you should endeavor not to manually edit or create connection objects and instead maintain an accurate site topology that the KCC can use to build and maintain the correct connection object topology. If the name of the connection object in the MMC doesn't display as "<automatically generated>," that means that Active Directory is not managing the connection object.

Knowledge Consistency Checker

In all but the smallest deployments, such as small business server (SBS) or other single-site environments, administrators should create a site topology in Active Directory that maps closely to the physical network. Unfortunately, Active Directory is not intelligent enough to look at the network and build its own complex topologies to optimize replication and resource location for every company installing Active Directory—it needs your help. Creating a site topology is discussed in Chapter 14.

Once you have set up your sites, subnets, and site link objects, an Active Directory process called the *Knowledge Consistency Checker* (KCC) takes that information and automatically generates and maintains the connection objects that describe which naming contexts should be replicated between which domain controllers, as well as how and when. The KCC has two separate algorithms it uses to determine what connection objects are needed: intrasite and intersite.

The intrasite algorithm is designed to create a minimal latency ring topology *for each naming context* that guarantees no more than three hops between any two domain controllers in the site. As DCs, GCs, and domains are added and removed within a site, the KCC adds and removes connections between the domain controllers as necessary to maintain this minimal-hop topology. It is quite simple to visualize when dealing with a single domain and a small number of domain controllers, but gets extremely difficult to visualize as you add many domain controllers and additional domains.

The intersite algorithm, on the other hand, is not a minimum-hop algorithm; it tries to keep the sites connected via a spanning-tree algorithm so that replication can occur, and then simply follows the site link metrics for making those connections. It is quite possible for the KCC to generate a replication topology that forces a change to replicate through eight sites to get from one end of the topology to the other. If you are unhappy with the site connections made by the KCC for the intersite topology because they don't align with your network connections, the problem is almost certainly related to how

the site links are configured. A well-designed site topology will help the KCC generate an intelligent and efficient collection of connection objects for replication.

We cover the KCC in greater depth in Chapter 14.

How Replication Works

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 your replication topology, you should understand the basics of how replication works, but you also need to understand how replication works using these new terms, which are used throughout both Microsoft's documentation and its management tools. As you read the rest of this chapter, refer back as needed to the definitions of the terms that are presented. Do not be disappointed if it doesn't all sink into your brain comfortably on your first or even fifth read of the material. Even experienced administrators have been known to debate how this all works, as well as what the proper terms are for the various structures and processes.

We will use the sample replication topology in Figure 6-11 for the discussion in this section.

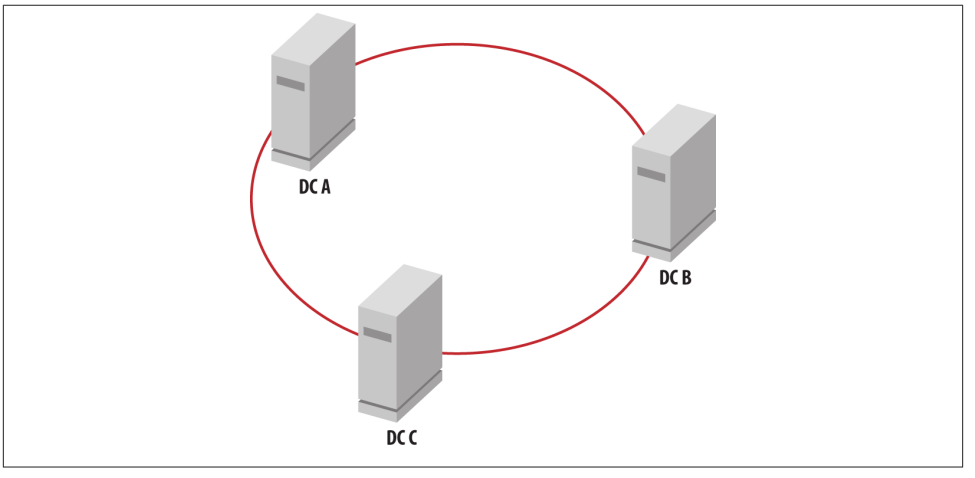
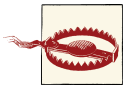


Figure 6-11. Sample replication topology for "How Replication Works" discussion

A Background to Metadata

Replication metadata is the data that governs the replication process. Active Directory replication enables data transfer between naming contexts on different domain controllers 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. That means that 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, and so forth. This is done this way because all replication is naming context-based. When a domain controller is pulling changes from another domain controller, it does so one naming context at a time in a serial fashion.



Although time isn't used as the basis for replication in Active Directory, it is still incredibly important. All authentication between domain controllers for replication is Kerberos-based, and Kerberos has a maximum time-skew requirement between hosts. In a domain with the default settings, if a domain controller deviates from the common time of all of the other domain controllers by more than five minutes, that domain controller will no longer be able to replicate with other domain controllers.

Update sequence numbers (USNs) and highestCommittedUSN

Each domain controller maintains its own separate u*pdate sequence number* (USN). Each time a change (new objects, updates, deletes, etc.) is made to an object in Active Directory, a transaction occurs. A USN is a 64-bit value that is assigned to each atomic update transaction, and it is meaningful only in relation to the domain controller on which the update was performed. Each separate update transaction will generate an incrementing USN value. A single USN could represent the change of every attribute on an object or a single attribute on a single object. If we examined a domain controller's highest committed USN and discovered it was 1,000 and then came back 10 minutes later and discovered the highest committed USN was now 1,056, we would know that 56 transactions had occurred in the past 20 minutes on this domain controller.

While some directory systems use timestamps to control replication and what needs to be propagated from server to server, Microsoft chose to use USN values. The fact that these values increment sequentially completely avoids the issues of the domain controllers having their clocks set backward or being out of sync with their replication partners. While the USN is independent of the clock on the domain controller, it is critical that the USN only move forward. When the USN on a domain controller gets decremented (usually through reverting a virtual machine snapshot), you will find yourself in a situation commonly referred to as *USN rollback*. Each domain controller maintains its highest combined USN for every naming context in the `highestCommittedUSN` value of the `RootDSE`. See the sidebar "Querying RootDSE with LDP" on page 64 in Chapter 4 for the steps to view the `highestCommittedUSN` value.

USNs are used to uniquely identify each update that has taken place in a naming context on a particular domain controller, regardless of the update type. It is highly improbable that the same USNs will ever represent the same change on two different domain

controllers. This means that you can request individual changes based on particular USNs from a specific domain controller, but it does not allow for direct comparison between DCs based on similar USN values. The change of the description on a computer account named *Cerberus* to "joe's computer" may be USN 23865 on one domain controller, but USN 4078 on another and USN 673459234 on yet another.

Originating updates versus replicated updates

Replication distinguishes between two types of update:

Originating update (write)
: This term defines the point of origin for a particular update—i.e., on which domain controller the change initially occurred.

Replicated update (write)
: This term defines the opposite of an originating update—i.e., the change in question did not begin here; it was replicated from another domain controller.

If you use the Active Directory Users and Computers snap-in to create five users on DC A, DC A's USN will be incremented five times, once for each originating update. These changes will then be replicated to DC B, incrementing its USN by five. Similarly, if DC A receives a change from DC B, DC A's USN is incremented once. This behavior is outlined in Table 6-1.

If an Active Directory database transaction is aborted—i.e., fails to complete—the associated USN value is not assigned to any object or reused in any way. The USN continues incrementing as changes occur.

Table 6-1. USN increment example

Action	Naming context	Originating domain controller	Domain controller	USN
Initial USN value	Domain NC	N/A	DC A	1000
Initial USN value	Domain NC	N/A	DC B	2500
Create 5 new users (originating update)	Domain NC	DC A	DC A	1005
Receive 5 new users (replicated update)	Domain NC	DC A	DC B	2505
Modify user description attribute (originating update)	Domain NC	DC B	DC B	2506
Receive modified user description attribute (replicated update)	Domain NC	DC B	DC A	1006

DSA GUIDs and invocation IDs

Each domain controller has a GUID called the Directory Service Agent (*DSA) GUID*. This DSA GUID is used to uniquely identify a domain controller and is the `object`

GUID of the NTDS Settings object viewable under the domain controller in Active Directory Sites and Services. This GUID does not change unless the domain controller is demoted and re-promoted.

The Active Directory database (*ntds.dit*) also has a GUID. This latter GUID is used to identify the server's Active Directory database in replication calls and is called the *invocation ID*; it is stored in the `invocationId` attribute of the NTDS Settings object for the domain controller. The invocation ID is changed any time Active Directory is restored on that DC or any time the DSA GUID is changed. Windows Server 2012 DCs will also reset their invocation IDs any time they detect a change in their virtual machine generation IDs. For more information on the virtual machine generation ID (VM gen ID) change process, refer to Chapter 9.

This change of GUID ensures that the other domain controllers on the network realize that this is a new database instance and that they create new high-watermark vector and up-to-dateness vector entries in their respective tables for it.

This allows changes that occurred after the backup to be replicated back to the newly restored database to bring it properly up to date. Without this step, any changes that originated on the restored DC after the backup completed and that had replicated out to the rest of the directory would never replicate back to this DC once it was restored. This is because the replication partners would assume the DC already had those changes, since it was the master of the changes in the first place.

Table 6-2 summarizes the DSA GUID and invocation ID values.

Table 6-2. DSA GUID and invocation ID summary

Object	Attribute	Description
NTDS Settings	`objectGUID`	Uniquely identifies the domain controller for its entire lifetime. This attribute value is referred to as the DSA GUID.
NTDS Settings	`invocationID`	Uniquely identifies the Active Directory database on the domain controller. This attribute value is referred to as the invocation ID. The invocation ID is reset when a domain controller is restored from a backup or when a change in VM gen ID is detected.

High-watermark vector (direct up-to-dateness vector)

The high-watermark vector (HWMV) is a table maintained independently by every domain controller to assist in efficient replication of a naming context. Specifically, it is used to help the domain controller determine where it last left off when replicating the naming context with a specific replication partner.

There is one table for every naming context the domain controller maintains a replica of, so at a minimum every domain controller will have at least three HWMV tables: one each for the Schema, the Configuration, and Domain NCs. Each table stores the highest USN of the updates the domain controller has received from each direct partner it replicates with for the given naming context. These USN values are used to determine where replication should begin with each partner on the next replication cycle. This allows the domain controller to request the most recent updates from a given replication partner for each naming context.

When the local domain controller initiates replication for a naming context with one of its partners, the highest USN for that partner from the domain controller's high-watermark vector for the naming context is one of the pieces of information sent to the replication partner. The replication partner compares that value with its current highest USN for the naming context to help determine what changes should be sent to the domain controller. This logic is further refined by the up-to-dateness vector, as described in the next section.

Continuing with our previous example, the HWMV tables of DC A and DC B are outlined in Table 6-3 and Table 6-4.

Table 6-3. DC A's high-watermark vector table

Naming context	Replication partner	High-watermark vector (USN of last update received)
Domain NC	DC B invocation ID	2506
Domain NC	DC C invocation ID	3606

Table 6-4. DC B's high-watermark vector table

Naming context	Replication partner	High-watermark vector (USN of last update received)
Domain NC	DC A invocation ID	1006
Domain NC	DC C invocation ID	3606

Up-to-dateness vector

The up-to-dateness vector (UTDV) is a table maintained independently by every domain controller to assist in efficient replication of a naming context. Specifically, it is used for replication dampening to reduce needless replication traffic and endless replication loops.

There is one table for every naming context the domain controller maintains a replica of, so again, at a minimum, every domain controller will have at least three of these tables. Each table stores the highest originating update USN the domain controller has received from every other domain controller that has ever existed in the forest, as well

as the date/time at which the domain controller last successfully completed a replication cycle with the given replication partner and naming context.

The up-to-dateness vector is used in conjunction with the high-watermark vector to reduce replication traffic. When the replication request for a naming context is passed to the replication partner, the destination domain controller's up-to-dateness vector for the naming context is also in the request. The source partner can then zero in on changes that it hasn't previously sent and then further filter out any changes that the destination may have already received from other replication partners. In this way, it guarantees that a single change is not replicated to the same domain controller multiple times; this is called *propagation dampening*.

Let's examine our sample replication scenario again; however, this time we will include DC C in the scenario. Table 6-5 shows the USN values as a result of each step.

Table 6-5. USN values including DC C

Action	Naming context	Originating domain controller	Domain controller	USN
Initial USN value	Domain NC	N/A	DC A	1000
Initial USN value	Domain NC	N/A	DC B	2500
Initial USN value	Domain NC	N/A	DC C	3750
Create 5 new users (originating update)	Domain NC	DC A	DC A	1005
Receive 5 new users (replicated update)	Domain NC	DC A	DC B	2505
Receive 5 new users (replicated update)	Domain NC	DC A	DC C	3755
Modify user description attribute (originating update)	Domain NC	DC B	DC B	2506
Receive modified user description attribute (replicated update)	Domain NC	DC B	DC A	1006
Receive modified user description attribute (replicated update)	Domain NC	DC B	DC C	3756

Once DC C receives the updates originated by DC A, it will attempt to replicate them to DC B; however, DC B has already received these updates from DC A directly. To prevent an endless replication loop, DC C checks the up-to-dateness vector tables for DC B, and it then knows not to send those updates.

Let's also look at the high-watermark vector tables again, now that DC C is included. Tables 6-6, 6-7, and 6-8 show the HWMV and UTDV tables for DC A, DC B, and DC C.

Table 6-6. DC A's high-watermark and up-to-dateness vector tables

Naming context	Replication partner	High-watermark vector (USN of last update)	Up-to-dateness vector
Domain NC	DC B invocation ID	2506	2506
Domain NC	DC C invocation ID	3756	3756

Table 6-7. DC B's high-watermark and up-to-dateness vector tables

Naming context	Replication partner	High-watermark vector (USN of last update)	Up-to-dateness vector
Domain NC	DC A invocation ID	1006	1006
Domain NC	DC C invocation ID	3756	3756

Table 6-8. DC C's high-watermark and up-to-dateness vector tables

Naming context	Replication partner	High-watermark vector (USN of last update)	Up-to-dateness vector
Domain NC	DC A invocation ID	1006	1006
Domain NC	DC B invocation ID	2506	2506

Notice that now each domain controller has an up-to-dateness vector for each domain controller that it replicates. When DC B goes to pull changes from DC C, it includes its up-to-dateness vector table in the request. DC C will then check this table and discover that DC B has already received the change from DC A. This functionality prevents the endless loop that would otherwise occur and also eliminates unnecessary replication traffic on the network.

Recap

The following list summarizes the important points of this section:

- Active Directory is split into separate naming contexts, each of which replicates independently.
- Within each naming context, a variety of metadata is held.

  For each naming context on a given domain controller, a high-watermark vector is maintained that contains one entry for each of its replication partners. The values in this table for the replication partners are updated only during a replication cycle.

  For each naming context on a given domain controller, an up-to-dateness vector is maintained that contains one entry for every domain controller that has ever made an originating write within this NC. Each entry consists of three values:

  — The originating server's DSA GUID

  — The orginating server's USN

  — A timestamp indicating the date and time of the last successful replication with the originating domain controller

These values are updated only during a replication cycle.

How an Object's Metadata Is Modified During Replication

To minimize the use of abbreviations, the terms *DC* and *server* are used interchangeably in this section. The terms *property* and *attribute* are also used interchangeably here.

To see how the actual data is modified during replication, consider a four-step example:

1. An object (a user) is created on DC A.
2. That object is replicated to DC B.
3. That object is subsequently modified on DC B.
4. The new changes to that object are replicated back to DC A.

This four-step process is shown in Figure 6-12. The diagram depicts the status of the user object on both DC A and DC B during the four time periods that represent each of the steps.

Now use Figure 6-12 to follow a discussion of each of the steps.

Step 1: Initial creation of a user on Server A

When you create a user on DC A, DC A is the originating server. During the Active Directory database transaction representing the creation of the new user on DC A, a USN (1000) is assigned to the transaction. The user's `uSNCreated` and `uSNChanged` attributes are automatically set to `1000` (the USN of the transaction corresponding to the user creation). All of the user's attributes are also initialized with a set of data, as follows:

- The attribute's value(s) is/are set according to system defaults or parameters given during user creation.
- The attribute's USN is set to 1000 (the USN of this transaction).
- The attribute's version number is set to 1.
- The attribute's timestamp is set to the time of the object creation.
- The attribute's originating-server GUID is set to the GUID of DC A.
- The attribute's originating-server USN is set to 1000 (the USN of this transaction).

This information tells you several things about the user:

- The user was created during transaction 1000 on this domain controller (`uSNCreated = 1000`).
- The user was last changed during transaction 1000 (`uSNChanged = 1000`).
- The attributes for the user have never been modified from their original values (property version numbers = 1), and these values were set during transaction 1000 (attributes' USNs = 1000).
- Each attribute was last set by the originating server, DC A, during transaction 1000 (originating-server GUID and originating-server USN).

The preceding discussion showed two per-object values and five per-attribute values being changed. While `uSNChanged` and `uSNCreated` are real attributes on each object in Active Directory, attributes of an object can only have values and cannot hold other attributes, such as a version number.

In reality, all of the per-attribute replication metadata (Property Version Number, Time-Changed, Originating-DC-GUID, Originating-USN, Property-USN) for every attribute of any given object is encoded together as a single byte string and stored as `replPropertyMetaData`, a nonreplicated attribute of the object.

Use PowerShell, *repadmin*, *adfind*, ADSI Edit, or LDP to see a property's metadata. For an example of how to view replication metadata, see the sidebar "Viewing Replication Metadata" on page 131.

Viewing Replication Metadata

Replication metadata is easily viewable with a number of tools and can be a key component of the troubleshooting process, particularly if you are trying to determine when and where data in Active Directory has changed. In this example, we will use *repadmin* to display the replication metadata of the Administrator account:

```
repadmin /showobjmeta "DC01" "CN=Administrator,CN=Users,DC=contoso,DC=com"
```

repadmin requires two arguments in this example:

- The domain controller to query the metadata on
- The distinguished name of the object to display metadata for

A partial version of the resulting output is shown below. You will notice that each replicated attribute stores the local and originating USNs, the originating domain controller, the date and time of the change, and the version number.

```
Loc.USN  Originating DSA Org.USN   Org.Time/Date        Ver Attribute
=======  =============== ========= ===================  === ============
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  objectClass
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  cn
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  description
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  instanceType
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  whenCreated
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  displayName
8194     SiteA\DC01      8194      2008-03-24 21:27:24   1  name
15987    SiteA\DC02      12117     2008-04-14 11:07:42   2  pwdLastSet
```

You can also use the *Get-ADReplicationAttributeMetadata* cmdlet to retrieve the same information with Windows PowerShell. Windows PowerShell provides a much stronger interface for working with data such as this. The following command (entered all on one line) sorts the data to match the output from *repadmin*:

```
Get-ADReplicationAttributeMetadata `
-Object "CN=Administrator,CN=Users,DC=contoso,DC=com" `
-Server dc01 | Select LocalChangeUsn, `
LastOriginatingChangeDirectoryServerIdentity, `
LastOriginatingChangeUsn, LastOriginatingChangeTime, `
Version, AttributeName | ft
```

Step 2: Replication of the originating write to DC B

Later, when this object is replicated to DC B, DC B adds the user to its copy of Active Directory as a replicated write. During this transaction, USN 2500 is allocated, and the user's `uSNCreated` and `uSNChanged` attributes are modified to correspond to DC B's transaction USN (2500).

From this we can learn several key points:

- The user was created during transaction 2500 on this server (`uSNCreated = 2500`).
- The user was last changed during transaction 2500 (`uSNChanged = 2500`).
- The attributes for the user have never been modified from their original values (property version numbers = 1), and these values were set during transaction 2500 (attributes' USNs = 2500).
- Each attribute was last set by the originating server, DC A, during transaction 1000 (originating-server GUID and originating-server USN).

Step 3: Password change for the user on DC B

Now an originating write (a password change) occurs on DC B's replicated-write user. Some time has passed since the user was originally created, so the USN assigned to the password-change transaction is 3777. When the password is changed, the user's `uSN`

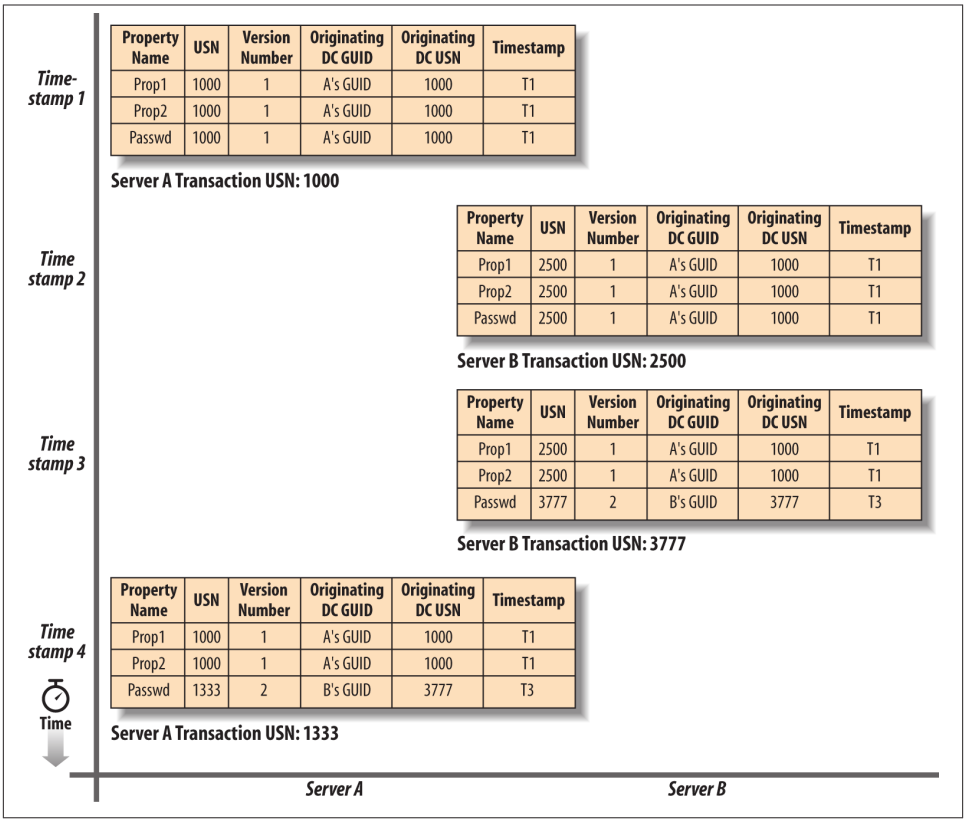


Figure 6-12. How metadata is modified during replication

`Changed` property is modified to become 3777. In addition, the password attribute (and only the password attribute) is modified in the following way:

- The password value is set.
- The attribute's USN is set to 3777 (the USN of this transaction).
- The attribute's version number is set to 2.
- The attribute's timestamp is set to the date/time at which transaction 3777 occurred.
- The attribute's originating-server GUID is set to the GUID of Server B.
- The attribute's originating-server USN is set to 3777 (the USN of this transaction).

Looking at the user object, you can now see that the object was last changed during transaction 3777 and that the transaction represented a password change that originated on DC B.

Step 4: Password-change replication to DC A

This step is similar to step 2. When DC A receives the password update during replication, it allocates the change transaction a USN of 1333.

Remember that updates occur at the attribute level and not the object level, so only the password is sent and not the entire user object.

During transaction 1333, the user's `uSNChanged` attribute is modified to correspond to Server A's transaction USN.

If you are duplicating this step-by-step walkthrough on real domain controllers, you will have noticed two discrepancies.

The first is that after the replication from DC A to DC B, the `cn` attribute will actually show that the originating write came from DC B. This is a backend implementation detail impacting the RDN attribute for any given object, and it is the only case where this type of discrepancy occurs.

The second is that a password change actually updates several attributes, not just one. The attributes involved are `dBCSPwd`, `unicodePwd`, `pwdLastSet`, `ntPwdHistory`, and `lmPwdHistory`. This is another backend implementation detail. For most attributes, when you specifically update a single attribute, only that attribute gets modified and replicated; passwords are handled as a special case.

From this, we can learn a number of things:

- The user was created during transaction 1000 on this server (`uSNCreated = 1000`).
- The user was last changed during transaction 1333 (`uSNChanged = 1333`).
- All but one of the attributes for the user have retained their original values (property version numbers = 1), and these values were set at transaction 1000 (properties' USNs = 1000).
- All but one of the attributes were last set by the originating server, DC A, during transaction 1000 (originating-server GUID and originating-server USN).
- The password was modified for the first time since its creation (password version number = 2) during transaction 1333 (password's USN = 1333), and it was modified on DC B during transaction 3777 (originating-server GUID and originating-server USN).

That's how object and property metadata is modified during replication. Let's now take a look at exactly how replication occurs.

The Replication of a Naming Context Between Two Servers

In the following examples, there are five servers in a domain: Server A, Server B, Server C, Server D, and Server E. It doesn't matter what NC they are replicating or which servers replicate with which other servers (as they do not all have to replicate with one another directly), because the replication process for any two servers will be the same nonetheless. Replication is a five-step process:

1. Replication with a partner is initiated.
2. The partner works out what updates to send.
3. The partner sends the updates to the initiating server.
4. The initiating server processes the updates.
5. The initiating server checks whether it is up to date.

Step 1: Replication with a partner is initiated

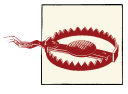
Replication occurs between only two servers at any time, so let's consider Server A and Server B, which are replication partners. At a certain point in time indicated by the replication schedule on Server A, Server A initiates replication for a particular NC with Server B and requests any updates that it doesn't have. This is a one-way update transfer from Server B to Server A. No new updates will be passed to Server B in this replication cycle, as this would require Server B to initiate the replication.

Server A initiates the replication by sending Server B a request to replicate along with five pieces of important replication metadata, (i.e., data relating to the replication process itself). The five pieces of metadata are:

- The name of the NC for which Server A wishes to receive updates
- The maximum number of object updates that Server A wishes to receive during this replication cycle
- The maximum number of values that Server A wishes to receive during this replication cycle
- The USN for Server B from Server A's high-watermark vector for this NC
- Server A's up-to-dateness vector for this NC

Specifying the numbers of maximum object updates and property values is very important in limiting network bandwidth consumption. If one server has had a large volume of updates since the last replication cycle, limiting the number of objects replicated

out in one replication packet means that network bandwidth is not inordinately taken up by replicating those objects in one huge packet. Instead, the replication is broken down into smaller packets over the course of the replication cycle. Once a replication cycle has started for a naming context, it will replicate all changes regardless of how many packets are needed or how long it will take, unless the connection between the domain controllers fails outright.



The wording in the previous paragraph with regard to the behavior of replication schedules is subtle but key. The schedule defines only when replication can *begin*. Once a replication cycle begins, it will not stop until it completes (or the connection is interrupted). This is true even if the schedule is overrun.

This step is illustrated in Figure 6-13, which shows that while the initiation of the replication occurs from an NC denoted as xxxx on Server A (where xxxx could represent the Schema NC, the Configuration NC, an application partition, or any domain NC), the actual replication will occur later from Server B to Server A.

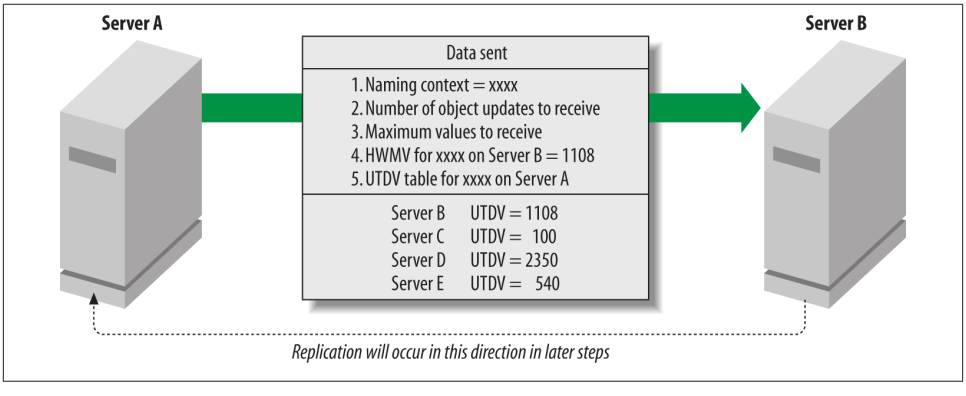


Figure 6-13. Initiating replication with Server B for NC xxxx

Step 2: The partner works out what updates to send

Server B receives all this metadata and works out which updates it needs to send back for this NC. First, Server B determines its own highest committed USN for its copy of the NC and compares that to the USN Server A submitted from its high-watermark vector table. Assuming that there have been some updates, Server B instantly knows how many updates have occurred since Server A last replicated with Server B. This has to be true, as Server A's HWMV will have been updated with Server B's highest committed USN for the NC during the last replication cycle. Any difference between the two USNs must therefore represent changes on Server B since the last replication, and Server B knows which individual USNs Server A is missing. Assuming also for now that

the number of updates does not exceed the maximums specified by Server A in its metadata, Server B can supply all of the missing updates to Server A.

However, this entire set of updates may not need to go to Server A if Server A has already had some of them replicated from other servers. Server B now needs some way of knowing which updates Server A has already seen, so that it can remove those items from the list of updates to send. That's where the up-to-dateness vector comes in. For each update that could potentially be sent, Server B checks two pieces of data attached to the object that was updated: the DSA GUID of the server that originated the update (the Originating-DC-GUID) and the USN associated with that update (the Originating-USN) on the originating server. For example, a password change to a user may have been replicated to Server B and recorded as USN 1112, but it may in fact have originated on Server D as USN 2345. Server B cross-references the originating server's GUID with Server A's UTDV to find the highest originating write USN for the originating server. If the USN recorded in the UTTV for the originating server is equal to or higher than the USN attached to the update on Server D, Server A must have already seen the update. This has to be true, because Server A's UTDV is used to indicate the highest originating writes that Server A has received.

Let's say that Server B has four updates for Server A: one originating write (Server B USN 1111) and three replicated writes (Server B USNs 1109, 1110, and 1112). The reason there are four is that 1112 is the last update made on Server B in this example, and the USN in Server A's HWMV for xxxx on Server B from Figure 6-13 is 1108. So, we look for updates starting at 1109 up to the last update on Server B, which is 1112. The first two replicated writes (Server B USNs 1109 and 1110) originated on Server E (Server E USNs 567 and 788), and one (Server B USN 1112) originated on Server D (Server D USN 2345). This is shown in Table 6-9.

Table 6-9. Potential updates to be sent

Server B USN	Originating-DC-GUID	Originating-DC-USN
1109	Server E's GUID	567
1110	Server E's GUID	788
1111	Server B's GUID	1111
1112	Server D's GUID	2345

According to Figure 6-13, Server A already has Server D's 2345 update, because the USN in Server A's UTDV for Server D is 2350. So, both Server A and Server B already have Server D's 2345 update, and there is no need to waste bandwidth sending it over the network again. (As mentioned earlier, the act of filtering previously seen updates to keep them from being continually sent between the servers is known as "propagation dampening.")

Now that you know how the high-watermark vector and up-to-dateness vector help Server B to work out what updates need to be sent, let's look at the exact process that Server B uses to work out what data is required.

When Server B receives a request for updates from Server A, the following steps occur:

1. Server B makes a copy of its up-to-dateness vector for Server A.
2. Server B puts the table to one side, so to speak, and does a search of the entire naming context for all objects with a `uSNChanged` value greater than the USN value from Server A's high-watermark vector entry for Server B. This list is then sorted into ascending `uSNChanged` order.
3. Server B initializes an empty output buffer to which it will add update entries for sending to Server A. It also initializes a value called Last-Object-USN-Changed. This will be used to represent the USN of the last object sent in this particular replication session. This value is not an attribute of any particular object, just a simple piece of replication metadata.
4. Server B enumerates the list of objects in ascending `uSNChanged` order and uses the following algorithm for each object:
   - If the object has already been added to the output buffer, Server B sets Last-Object-USN-Changed to the `uSNChanged` property of the current object. Enumeration continues with the next object.
   - If the object has not already been added to the output buffer, Server B tests the object to see if it contains changes that need to be sent to the destination. For each property of the current object, Server B takes the Originating-DC-GUID of that property and locates the USN that corresponds to that GUID from Server A's UTDV. From that entry, Server B looks at the Originating-USN. If the property's Originating-USN on Server B is greater than Server A's UTDV entry, the property needs to be sent.
   - If changes need to be sent, an update entry is added to the output buffer. Server B sets Last-Object-USN-Changed to the `uSNChanged` property of the current object. Enumeration continues with the next object.
   - If no changes need to be sent, Server B sets Last-Object-USN-Changed to the `uSNChanged` of the current object. Enumeration continues with the next object.

During the enumeration, if the requested limit on object update entries or values is reached, the enumeration terminates early and a flag known as `More-Data` is set to `True`. If the enumeration finishes without either limit being hit, `More-Data` is set to `False`.

Step 3: The partner sends the updates to the initiating server

Server B identifies the list of updates that it should send back based on those that Server A has not yet seen from other sources. Server B then sends this data to Server A. In addition, if `More-Data` is set to `False`, one extra piece of metadata is sent back. The returned information from Server B includes:

- The output buffer updates from Server B
- Server B's Last-Object-USN-Changed value (i.e., the value for Server A to insert into the high-watermark vector for the NC for Server B)
- The `More-Data` flag
- Server B's up-to-dateness vector for this NC (sent only when `More-Data` is set to `False`)

This is shown in Figure 6-14.

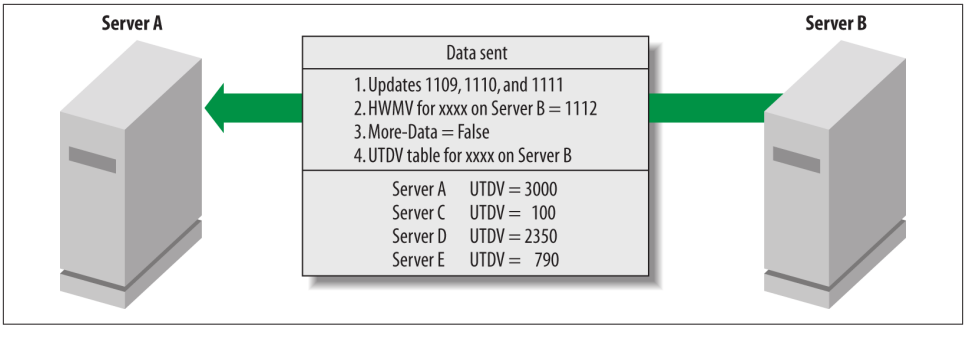


Figure 6-14. Server B sends the updates to Server A for NC xxxx

If Server B calculates that Server A is already up to date and requires no updates, only the last two pieces of metadata are returned to Server A. This can occur if the highest committed USN for the NC on Server B is identical to the HWMV entry passed by Server A (i.e., no updates have occurred since the last replication cycle), or if Server B's highest committed USN for the NC has changed, but Server A has already seen all of the originating updates through replication with other partners. In both cases, just the metadata is returned.

Step 4: The initiating server processes the updates

Server A receives the data. For each update entry it receives, Server A allocates a USN and starts a database transaction to update the relevant object in its own copy of the Active Directory database. The object's `uSNChanged` property is set to the USN of this transaction. The database transaction is then committed. This process continues for each update entry that was received.

After all the update entries have been processed, the USN in Server A's high-watermark vector for Server B is set to the Last-Object-USN-Changed received from Server B. In other words, Server A now knows that it is up to date with Server B, up to the last change just sent over.

The Last-Object-USN-Changed that Server A receives allows it to know the last update that Server B has made. This will be used in the next replication cycle. In the previous example, the highest update sent across to Server A is USN 1111. Server B's USN 1112 update is not actually sent since Server A has already seen it. However, the Last-Object-USN-Changed returned by Server B with the data would still be 1112 and not 1111.

Step 5: The initiating server checks whether it is up to date

Server A now checks the `More-Data` flag. If `More-Data` is set to `True`, Server A goes back to step 1 to start replication with Server B again and request more updates. If `More-Data` is set to `False`, every update must have been received from Server B, and finally Server A's up-to-dateness vector is itself updated.

The up-to-dateness vector allows Server A to identify which originating updates Server B has seen and thus, by replication, which originating updates it has now seen. Server A does not replace its up-to-dateness vector with the one it was sent. Instead, it checks each entry in the received table and does one of two things. If the entry for a server is not listed in its own UTDV, it adds that entry to the table. This allows Server A to know that it has now been updated to a certain level for a new server. If the entry for a server is listed in Server A's UTDV and the value received is higher, it modifies its own copy of the table with the higher value. After all, it has now been updated to this new level by that server, so it had better record that fact.

Table 6-10 shows Server A's up-to-dateness vector and high-watermark vector for the xxxx naming context before step 1 and after step 5.

Table 6-10. State of UTDV and HWMV for Server A before and after updates

	HWMV for Server B	Server B UTDV	Server C UTDV	Server D UTDV	Server E UTDV
Before step 1	1108	1108	100	2350	540
After step 5	1112	1111	100	2350	790

Recap

The following points summarize replication between naming contexts:

- The high-watermark vector is used to detect updates that need to be sent between replication partners.
- The up-to-dateness vector is used in propagation dampening to filter the updates so that only updates that the initiating server has not seen are transmitted from a partner.

- The `uSNChanged` property on each object is used to identify which objects might need to be sent out as updates to the initiating server.

You can force manual replication of a particular NC on a DC if you choose, using the AD Sites and Services snap-in. Browse to the connection object that you want to replicate over, right-click it, and select Replicate Now. This can also be done from the command line with the *repadmin* utility.

With Windows PowerShell, you can use the *Sync-ADObject* cmdlet to force replication of a specific object. There is not, however, a cmdlet to synchronize an entire partition at once.

How Replication Conflicts Are Reconciled

While the replication process is usually fine on its own, there are times when conflicts can occur because two servers perform irreconcilable operations between replication cycles. For example, say Server A creates an object with a particular name at roughly the same time that Server B creates an object with the same name under the same parent container. Both can't exist at the same time in Active Directory, so what happens to the two objects? Does one object get deleted or renamed? Do both objects get deleted or renamed?

What about an administrator moving an object on Server D to an organizational unit while at the same time on Server B that organizational unit is being deleted? What happens to the soon-to-be orphaned object? Is it deleted along with the organizational unit or moved somewhere else entirely?

Consider a final example: if an admin on Server B changes a user's password while the user himself changes his password on Server C, which password does the user get?

All of these conflicts need to be resolved within Active Directory during the next replication cycle. The exact reconciliation process and how the final decision is replicated back out depend on the exact conflict that occurred.

Conflict due to identical attribute change

This scenario occurs when an attribute on the same object is updated on two different domain controllers at around the same time. During replication, each domain controller follows the following process to resolve the conflict:

1. The server starts reconciliation by looking at the version numbers of the two attributes. Whichever attribute has the higher version number wins the conflict.

2. If the property version numbers are equal, the server checks the timestamps of both attributes. Whichever attribute was changed at the later time wins the conflict.
3. If the attribute timestamps are equal, the GUIDs from the two originating servers are checked. As GUIDs must be unique, these two values have to be unique, so the server takes the attribute change from the originating server with the mathematically higher GUID as the winner of the conflict.

All replication metadata timestamps for Active Directory are stored in Universal Time Coordinated (UTC), which is more commonly known as Greenwich Mean Time (GMT) or Zulu time.

Conflict due to a move or creation of an object under a now-deleted parent

This scenario occurs when an object is created on one server under a particular OU or container—for example, an OU called People—and meanwhile, on another server at about the same time, that OU is deleted.

This is a fairly easy conflict to resolve. In this case, the parent (the People OU) remains deleted, but the object is moved to the naming context's *Lost and Found* container, which was specially set up for this scenario. The distinguished name of the Lost and Found container for the *mycorp.com* domain is:

```
cn=LostAndFound,dc=mycorp,dc=com
```

In addition to the Lost and Found container for the domain, every forest has a Lost and Found container for the configuration partition. For the *mycorp.com* forest, this would be found at:

```
cn=LostAndFoundConfig,cn=Configuration,dc=mycorp,dc=com
```

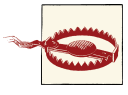
Conflict due to creation of objects with names that conflict

Recall that the relative distinguished name (RDN) for an object must be unique within the context of its parent container. This particular conflict occurs when two objects are created on two different servers with the same RDN. During replication, domain controllers take the following steps to resolve this conflict:

1. The server starts reconciliation by looking at the version numbers of the two objects. Whichever object has the higher version number wins the conflict.
2. If the object version numbers are equal, the server checks the timestamps of both objects. Whichever object was changed at the later time wins the conflict.

3. If both object timestamps are equal, the GUIDs from the two originating servers are checked. The server once again takes the object change from the originating server with the higher GUID as the winner of the conflict.

In this case, however, the object that lost the conflict resolution is not lost or deleted; instead, the conflicting attribute is modified with a known unique value. That way, at the end of the resolution, both objects exist, with one having its conflicting name changed to a unique value. The unique name has the following format: *<ObjectName><LineFeed>CNF:<ObjectGUID>*.



This logic is only used for the RDN naming value of the object (usually the `cn` attribute). It is not used for cleaning up other values that are normally required to be unique, such as the `sAMAccountName` attribute that represents a user's username or a computer's name. For more information on how duplicate `sAMAccountName` values are handled, see the upcoming sidebar "How Active Directory Handles Duplicate Account Values" on page 143.

In the following example, a computer called *PC04* has been renamed due to a conflict. This probably occurred after an administrator encountered an error joining the machine to the domain, and then subsequently tried again. During the subsequent try, the administrator connected to another domain controller before replication had occurred:

```
CN=PC04\0ACNF:41968ad2-717e-4ffc-a876-04798d288ede,CN=Computers,DC=contoso,DC=com
```

How Active Directory Handles Duplicate Account Values

While the replication engine handles duplicate RDNs by mangling the RDN with the object's GUID, Active Directory's process for handling duplicate `sAMAccountName` and `objectSID` security identifier (SID) values is totally separate. The detection of these types of duplicates occurs during various operations (such as logon or object name resolution) when the domain controller searches for an object and discovers multiple matches.

First, let's look at duplicate `sAMAccountName` values. When a duplicate user or computer `sAMAccountName` is found, the newest object is left as-is. Any others have their `sAMAccountName` values changed to the pattern of `$DUPLICATE-<RID>`, where *<RID>* is the relative identifier (RID) component of the object's SID.

The second scenario is duplicate `objectSID` values. This should never happen since the system generates the SID, but a domain controller rollback scenario, could, for example, introduce duplicate SIDs since RIDs would be reused. When a domain controller encounters a duplicate SID, all of the users or computers with that SID are immediately deleted from the domain.

In both scenarios, an event is logged to the System event log of the domain controller that discovers and resolves the duplicate. For information about these events, refer to these links:

- *http://technet.microsoft.com/en-us/library/ee410964(v=ws.10).aspx*
- *http://technet.microsoft.com/en-us/library/ee410964(v=ws.10).aspx*

Replicating the conflict resolution

Let's say that Server A starts a replication cycle. First, it requests changes from Server B and receives updates. Then Server A requests changes from Server C and receives updates. However, as Server A is applying Server C's updates in order, it determines that a conflict has occurred between these updates and the updates recently applied by Server B. Server A resolves the conflict according to the preceding guidelines and finds in Server C's favor. Now, while Server A and Server C are correct, Server B still needs to be updated with Server C's value.

To do this, when Server B next requests updates from Server A, it receives (among others) the update that originated on Server C. Server B then applies the updates it receives in sequence, and when it gets to the update that originated on Server C, it detects the same conflict. Server B then goes through the same conflict resolution procedure that Server A did and comes to the same result. So, it modifies its own copy of the object to accommodate the change.

Additional problems can occur when changes are made on a server and it goes down prior to replicating the changes. If the server never comes back up to replicate changes, those changes are obviously lost.

Alternatively, if the server comes back up much later and attempts to replicate those changes back to Active Directory, there is a much greater chance of conflict resolution (with that server failing the conflict, if many of the changes that were made on that server have subsequently been made in Active Directory more recently on other servers). This isn't a problem, but is something you need to be aware of.

Common Replication Problems

There are a couple of issues that often crop up in the field or in discussions of Active Directory replication. Neither of these are issues that you will want to run into, but, it's important to be aware of them so you can recognize the symptoms and be prepared to respond.

Lingering Objects

Lingering objects appear when a domain controller that has been offline for longer than the tombstone lifetime of your forest comes back online and begins replicating again. We'll talk about the concept of tombstone lifetime in detail in Chapter 18, but in a nutshell, it's how long a record of deleted objects is preserved in the Active Directory database to ensure deletions are replicated. Depending on when your forest was created, your tombstone lifetime will likely be either 60 days or 180 days, assuming it hasn't been modified from the default.

When a domain controller is offline for longer than the tombstone lifetime and is then reconnected, it will never receive notification of objects that were deleted while it was offline. As a result, those objects will not exist on any domain controller in the forest except the one that was brought back online. This can pose a problem in a number of ways. First, a user account that has been deleted (perhaps because the employee was terminated) will be available for use again if the specific domain controller is targeted. Next, old objects such as users and contacts that display in the global address list (GAL) if you're using Exchange might start appearing again for certain users who get their view of the GAL from the domain controller in question. Printers published to Active Directory are another common example. Print queues are published and removed from the directory all the time as a matter of normal operation. End users searching AD for a printer in the site containing the domain controller with lingering objects may find themselves attempting to connect to printers that no longer exist.

In addition to these scenarios, you will often see replication-related event log entries on other domain controllers that make reference to the possibility of lingering objects in the forest. There are a couple of steps you can take to control the propagation of lingering objects, as well as to clean up a domain controller that has them.

First, enable *strict replication consistency* on all of your domain controllers. This setting causes a domain controller to quarantine a replication partner if it detects that an attempt has been made to replicate a lingering object. When a lingering object is replicated back into the forest, the deletion is essentially undone, and the object is restored to the state it is in on the domain controller that has the lingering object. This behavior is undesirable. To enable strict replication consistency, set this registry setting:

```
Path: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters
Value Name: Strict Replication Consistency
Value Type: REG_DWORD
Value: 1
```

You can use *repadmin* to enable strict replication consistency on every DC in your forest. To do this, run this command: *repadmin /regkey * +strict*.

To clean up lingering objects on a domain controller, the easiest solution is to simply demote the domain controller and then re-promote it. If you can't do this, you can take advantage of the lingering object cleanup functionality in *repadmin*. You'll need to choose a domain controller that will be the source of truth. Active Directory will check that each object on the domain controller you are cleaning up also exists on your "source of truth" DC.

To check *DC17*'s domain partition (*contoso.com*) for lingering objects against *DC05*, follow these steps:

1. Resolve the DSA GUID (boldfaced) of *DC05* by running: *repadmin /showrepl DC05*. This produces output like the following:

   ```
   SiteA\DC05
   DSA Options: IS_GC
   Site Options: (none)
   DSA object GUID: 67c1ee19-4706-4c56-b1cb-3a299541376e
   DSA invocationID: 54770948-f0d3-44ed-ae26-291ab6200ed4
   ```

2. Run the "remove lingering objects command" on *DC17* in advisory mode to get a list of objects that will be removed:

   ```
   repadmin /removelingeringobjects DC17 67c1ee19-4706-4c56-b1cb-3a299541376e
   dc=contoso,dc=com /advisory_mode
   ```

   There are three arguments in the preceding command: the DC to check (*DC17*), the source DC's DSA GUID (*67c1...*), and the DN of the partition to validate (*dc=contoso,dc=com*).

3. Review the events logged in *DC17*'s event log (events 1938, 1946, and 1942) for a list of objects that will be removed.

4. Run the remove lingering objects command again without the */advisory_mode* switch to delete the lingering objects from *DC17*'s domain partition:

   ```
   repadmin /removelingeringobjects DC17 67c1ee19-4706-4c56-b1cb-3a299541376e
   dc=contoso,dc=com
   ```

You will need to repeat these steps for each naming context that *DC17* is hosting. This includes the Configuration NC, DNS NCs, and any Global Catalog partitions.

USN Rollback

As virtualization has become commonplace in IT, the concept of *USN rollback* has grown as a discussion topic and fear amongst many Active Directory administrators. USN rollback is a precarious situation that can lead to replication black holes and duplicate SIDs, amongst other issues. The most common avenues for running into USN rollback are rolling back virtual machine snapshots and the use of image-based disk backup programs. Microsoft has taken steps in every version of Active Directory to try to mitigate the impact of these operations, and Windows Server 2012 finally addresses

many angles of this problem head-on. We'll talk about the improvements Windows Server 2012 makes in Chapter 9, but for now we'll focus on the impact on replication without consideration of those specific improvements.

As we discussed earlier, the USN should only ever move forward. Each time a domain controller starts a transaction, it increments the local USN counter so that the transaction can be identified. Another way to think of the USN is as a logical clock. If you think about a clock, the clock never moves backward. Consider a virtualized domain controller with its USN similar to Figure 6-15:

1. At USN 5, an administrator takes a snapshot of the virtual machine (VM).
2. A number of changes are made locally, and the USN advances to 9.
3. Next, the administrator rolls back the snapshot. The USN is now 5.
4. A number of changes are made once again, and they are assigned USNs 6 to 9.

Step 4 is the key here. The administrator has made changes that have the same USNs as previous changes. This leads to an inability for other DCs to keep track in their HWMV tables.

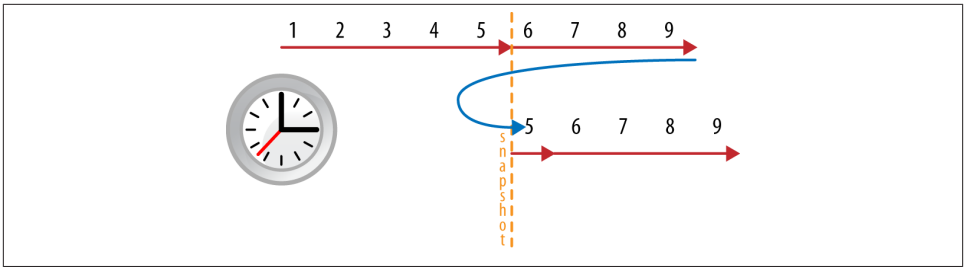


Figure 6-15. Active Directory's logical clock—the USN

To put this into better perspective, recall how we discussed earlier that DCs keep track of the changes they've replicated from other DCs in their high-watermark vector tables. Let's examine the previous scenario once more, factoring in the replication process as shown in Figure 6-16:

1. At T1, DC-A's `highestCommittedUSN` is `100`. DC-B's HWMV reflects this and the two DCs are in sync.
2. At T2, the administrator takes a snapshot of DC-A in the hypervisor.
3. At T3, the administrator creates 75 users on DC-A, incrementing DC-A's `highestCommittedUSN` to `175`. DC-B replicates these changes and updates its HWMV to reflect that it has replicated up to change 175 from DC-A.

4. At T4, the administrator rolls DC-A back to its state prior to taking the snapshot. DC-A's `highestCommittedUSN` is now `100` once again.
5. At T5, the administrator creates 15 users on DC-A, incrementing DC-A's `highestCommittedUSN` to 115. DC-B starts a replication cycle with DC-A by checking its HWMV and asking DC-A for any changes after USN 175. DC-A replies that it has no changes to send.

The net result here after T5 is that a set of objects exist on DC-A that will never replicate to DC-B. Until DC-A reaches USN 175, *any* changes made on DC-A will *never* replicate to DC-B. Since Windows Server 2003, Active Directory has attempted to detect when this scenario happens and quarantine replication so that the domain controller can be forcibly demoted and then re-promoted in order to get it back in sync with the rest of the forest. The good news is that, beginning with Windows Server 2012, virtualized DCs running on a hypervisor that provides a VM generation ID to the VM will not suffer from any rollback scenario that occurs within the state model of the hypervisor. Any "manual" snapshots that occur outside of the hypervisor's knowledge can still produce a rollback scenario, though, such as reverting the underlying files in the disk subsystem.

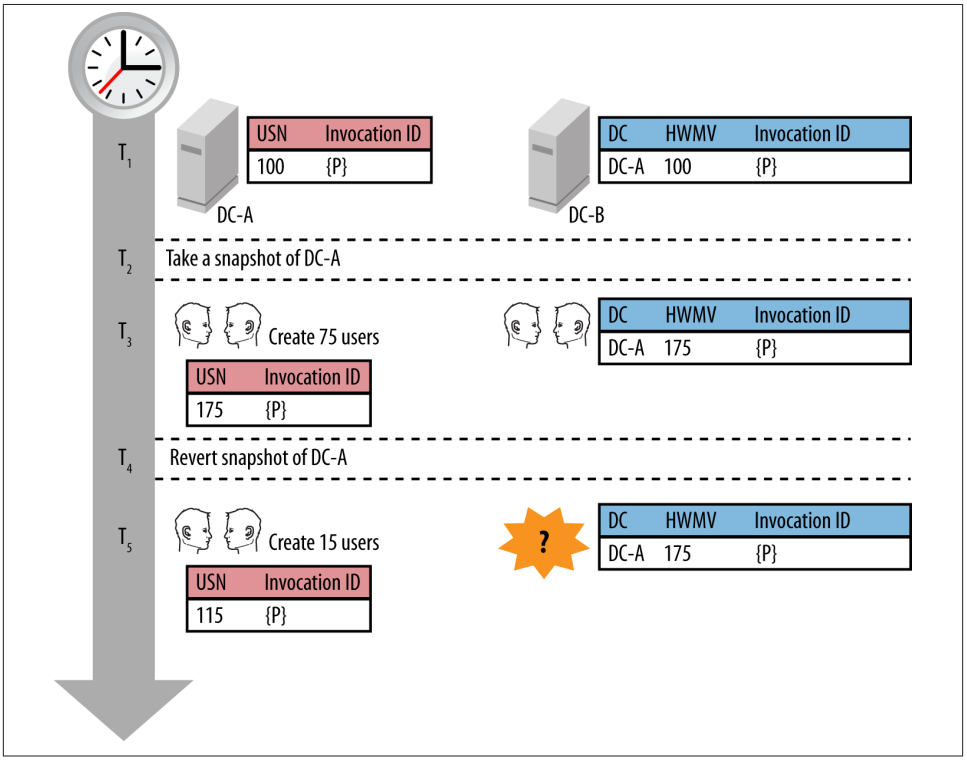


Figure 6-16. Active Directory replication with USN rollback

Summary

In this chapter, we looked at the importance of the site topology in Active Directory and how it relates to your physical network. We 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 resolved.

In Chapter 14, we'll 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 7

Searching Active Directory

The data stored in Active Directory is often crucial to providing accurate reports and powering business-critical applications. By querying the directory, you can gather data that meets your requirements directly, whether it's for a one-off report, a recurring process, or in support of an application that integrates with AD. In order to accomplish this, you need to be proficient in developing *LDAP filters* for your searches.

LDAP filters are relatively straightforward to build and understand, as the syntax is simple and the operators are very limited in quantity. But in addition to simply constructing an LDAP filter, being able to optimize that query so that it performs efficiently and doesn't have a negative impact on the performance of AD is a critical skill. Likewise, being able to analyze an application's query and make changes to the schema in order to improve performance is a fundamental troubleshooting skill.

The Directory Information Tree

Active Directory stores its database on each domain controller in the *ntds.dit* file, often simply referred to as "the DIT." DIT is short for *directory information tree*—a fancy moniker for the AD database. The structure of the data in the DIT is surprisingly simple, and if you come from a database background where you're familiar with optimizing data into many tables and iterations of normal forms, you'll probably be shocked at how AD stores its data.

Database Structure

Fundamentally, the DIT is organized into three key tables:

- The data table
- The link table

- The hidden table

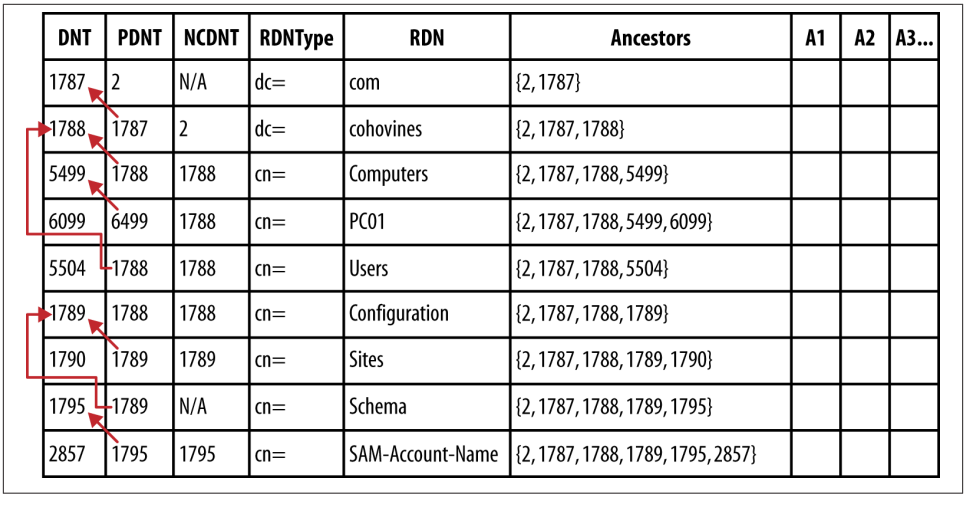
In addition, Windows Server 2003 and newer domain controllers also have a couple of tables that are used for single-instance storage of ACLs (the security descriptor table) and to track object quotas.

Hidden table

The hidden table is a single-row table Active Directory uses at startup to find configuration-related information in the data table. Namely, the hidden table contains a pointer to the domain controller's NTDS Settings object in the data table. In addition, there are a few other configuration-related items stored here.

Data table

The data table holds the bulk of the data in the Active Directory database. Regardless of its class (e.g., user versus organizational unit), each object in the directory, including the schema, is stored in an individual row in this table. Each attribute defined in the schema comprises a column in the data table, similar to Figure 7-1.



DNT	PDNT	NCDNT	RDNType	RDN	Ancestors	A1	A2	A3...
1787	2	N/A	dc=	com	{2,1787}			
1788	1787	2	dc=	cohovines	{2,1787,1788}			
5499	1788	1788	cn=	Computers	{2,1787,1788,5499}			
6099	6499	1788	cn=	PC01	{2,1787,1788,5499,6099}			
5504	1788	1788	cn=	Users	{2,1787,1788,5504}			
1789	1788	1788	cn=	Configuration	{2,1787,1788,1789}			
1790	1789	1789	cn=	Sites	{2,1787,1788,1789,1790}			
1795	1789	N/A	cn=	Schema	{2,1787,1788,1789,1795}			
2857	1795	1795	cn=	SAM-Account-Name	{2,1787,1788,1789,1795,2857}			

Figure 7-1. The data table

You might think there would be a storage penalty involved in having attribute columns with null values for objects that don't have that attribute defined in the schema. Fortunately, the database engine Active Directory uses, the Extensible Storage Engine (ESE), is very efficient at storing null values, and thus there isn't a penalty for storing data this way.

In addition to identifiers you often see in AD, such as SIDs for security principals or the `objectGUID` of every object in the directory, the data table also uses a special unique identifier for each row in the table. This identifier is known as a *distinguished name tag* (DNT) and is shown in the first column of Figure 7-1. When a new object is created, the next available DNT is assigned. An object's DNT is not replicated, so it is quite likely that the same object will have a different DNT on each domain controller. Additionally, DNTs are not reused when an object is deleted.

AD has a maximum of 2,147,483,393 ($2^{31} - 255$) DNTs in a given database. It is thus feasible that it will no longer be possible to create new objects on a given DC if it has stored this number of objects throughout its lifetime.

In addition to the DNT column, there are a number of references shown in Figure 7-1. The PDNT column stores the *parent DNT*. This is a pointer to the object's direct parent row in the database. When you move an object in the directory, its PDNT column entry is updated to match the object's new parent. The NCDNT column contains the DNT of the naming context head matching the NC in which the object resides. This NCDNT column helps to demonstrate that application partitions are simply logical partitions. You'll see that the Configuration and Schema naming contexts are stored alongside the Domain naming context in Figure 7-1. The NCDNT entry for objects in those NCs simply points to the relevant object's DNT.

Figure 7-1 also includes an Ancestors column. The Ancestors column is used during searching and is simply a listing of all of the DNTs between the root of the database and that object. In other words, the Ancestors column represents the hierarchy in the domain. Like the PDNT column, when an object is moved, the Ancestors column is updated to match.

As we mentioned, each attribute's data is stored as a separate column in the database, as represented by columns A1 through A3 and so forth. One detail to note is that the RDNType column actually stores the DNT of the RDN attribute in the schema for the object. We have removed this layer of abstraction to make it easier to follow the figures.

Link table

The next table—the link table, shown in Figure 7-2—is responsible for storing the data stored in linked attributes. Group membership is one common example of linked attribute data. Links are stored by referencing the DNT of the object with the forward link, the DNT of the object being linked to, and the link ID of the forward link attribute in the schema. In addition, there are a number of other columns not shown here that track replication metadata for linked value replication (LVR), track whether or not a

link is deactivated when the Recycle Bin is in use, and store optional link data for attributes that use DN-Binary or DN-String syntaxes.

Backlink_DNT	Link_DNT	LinkBase	Link_NCDNT
9601	5615	2	1788

Figure 7-2. The link table

Technically, the value in the LinkBase column is not a direct pointer to the forward link ID value for the attribute. We have simplified this detail in the figures as it is not relevant to the discussion.

The depiction of the link table in Figure 7-3 makes explaining the contents a bit easier. In this case, we can see that the row shown is tracking a membership in the Domain Admins group. Specifically, user Brian Desmond (DNT 9601) is a member of Domain Admins (DNT 5615). The LinkBase column tells us that the corresponding attribute has a link ID of 2.

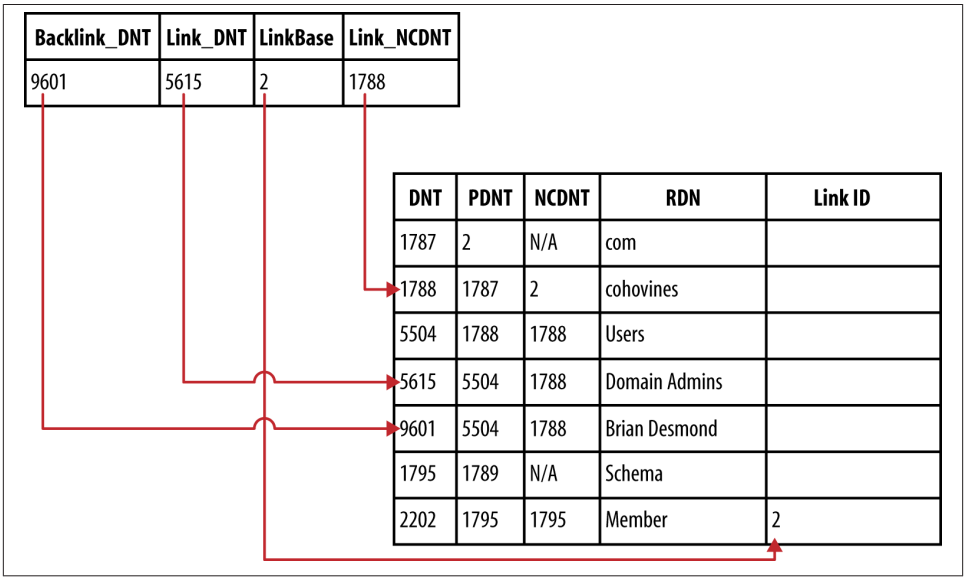


Figure 7-3. Link table references to the data table

Security descriptor table

Finally, the security descriptor table is used to deliver single-instance storage of access control lists (ACLs) in AD. This table was added in Windows Server 2003 to reduce the size of the AD database, since many organizations were finding that a large proportion of their AD databases were taken up by duplicate ACLs. The table works by storing the values for the `ntSecurityDescriptor` attribute as well as checksums for those values. The data table stores a pointer to the ACL (pointing to the row in the security descriptor table) for each object so the ACL only has to be stored once for objects that share the same ACL.

Searching the Database

The AD database is generally searched using an LDAP filter. The LDAP filter you supply to AD is transformed by the query processor into the necessary commands to actually find and retrieve records from the DIT. The language used to construct an LDAP filter is surprisingly simple, and in this section we'll look at what it takes to get data out of the directory.

You can use any number of tools to actually conduct your search. We recommend AdFind as well as LDP, which is included with Windows. We introduced LDP in Chapter 3, so we'll assume that you've taken a few minutes to review Chapter 3 before working with LDP here.

Filter Operators

The most basic query simply asks AD to return all of the objects that have an attribute containing a specific value. For example, if you wanted to return all of the objects in the Accounting Department, you would use this filter: `(department=Accounting)`. Here we've specified an attribute name of `department`, and a value of `Accounting`. There are a number of other operators you can take advantage of in addition to the equality (`=`) operator. Commonly used operators are shown in Table 7-1.

Table 7-1. LDAP filter operators

Operator	Description
=	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!	Not (negation)

One situation where you might want to take advantage of the greater than or equal to operator shown in Table 7-1 is when you're searching for locked-out accounts. You could construct a filter such as `(pwdLastSet>=1)` to find objects that have set their passwords

at least once. The negation or not operator is useful for finding a set of objects that do not match a given criterion. For example, if you want to find all the objects that are not in the Accounting Department, you could search for `(!(department=Accounting))`.

Using the `!` operator requires that Active Directory scan the entire database for objects satisfying the filter, even if the attribute being searched (such as `department`) is indexed. You should take this limitation into consideration when developing queries in order to ensure optimal performance.

In addition to the operators listed in Table 7-1, you can use the `*` character to ask for any object that has a specific attribute with a value. For example, the filter `(description=*)` will return all objects in the directory that have a value for the `description` attribute. Conversely, the filter `(!(description=*))` will return any object that does not have a `description` attribute populated with a value.

You can also use the `*` character to perform wildcard matches and medial searches. In the case of a wildcard search, you could conduct a search for `(givenName=a*)` to find all objects whose first name begins with the letter *A*. If you wanted to find all of the objects whose job title includes the string *vice* (such as vice president), you could do a search for `(title=*vice*)`. This is called a medial search.

Connecting Filter Components

The examples we've looked at so far are trivial in nature in that they only search a single attribute for a single value or criterion. Realistically, it is unlikely that you will commonly have such a trivial requirement. Instead, you'll likely need to combine multiple filters together to form a Boolean AND or Boolean OR expression (or both). Table 7-2 shows the operators you'll use to accomplish this.

Table 7-2. Boolean operators

Operator	Description
&	Boolean AND
|	Boolean OR

In the previous section, we searched for all of the objects in the Accounting Department. In addition to users, the `(department=Accounting)` filter will return other types of objects that have this `department` attribute value, such as contacts and groups. In order to scope the search to only return users in the Accounting Department, we need to specify the object class we want to filter on. To do this, we can use a filter such as `(&(objectClass=user)(department=Accounting))`. Here, we've asked Active Directory for objects of class `user` and with a `department` value of `Accounting`.

We can use the OR operator in the same fashion. If, for example, you want to find objects affiliated with either the Accounting or Marketing Departments, you could conduct a search for `(|(department=Accounting)(department=Marketing))`.

Finally, both the AND and OR operators can be used if you wanted to limit your search to users who work in either Accounting or Marketing. To do this, you could use a filter like this: `(&(objectClass=user)(|(department=Accounting)(department=Marketing))`. This may be a little difficult to follow when reading in a linear fashion. One trick for working with complex LDAP filters is to expand them into multiple indented lines, similar to this example:

```
(&
    (objectClass=user)
    (|
          (department=Accounting)
          (department=Marketing)
    )
)
```

Here we've constructed the same filter, but we've made it substantially easier to read by placing each component on a separate line and then indenting the expressions. If you have a complex query and you're looking for a quick way to arrange it into a more readable format, you can use AdFind. To format the previous query, you would run this command:

```
adfind -filterbreakdown
 (&(objectClass=user)(|(department=Accounting)(department=Marketing))
```

Just to give you an example of another complex query that stacks AND and OR operators together along with the NOT operator, consider this next example:

```
(|
    (&
          (objectClass=user)
          (|
               (department=Sales)
               (department=Marketing)
          )
          (!
               (employeeType=Contractor)
          )
    )
    (&
          (objectClass=contact)
          (|
               (company=Coho Vines)
               (company=Tailspin Toys)
          )
    )
)
```

In this query, we ask Active Directory to return objects that meet the following criteria:

- Users who are in the Sales or Marketing Departments.
- Contact objects that are from the Coho Vines or Tailspin Toys companies.

For reference, written on one line, this query is `(|(&(objectClass=user)(|(department=Sales)(department=Marketing))(!(employeeType=Contractor))(&(objectClass=contact)(|(company=Coho Vines)(company=Tailspin Toys))))`.

Search Bases

You can specify a *search base* in an LDAP search in order to tell Active Directory where in the hierarchy to begin searching. There are three types of search scopes: base, one-level, and subtree, as shown in Figure 7-4. By default, you will generally perform subtree searches.

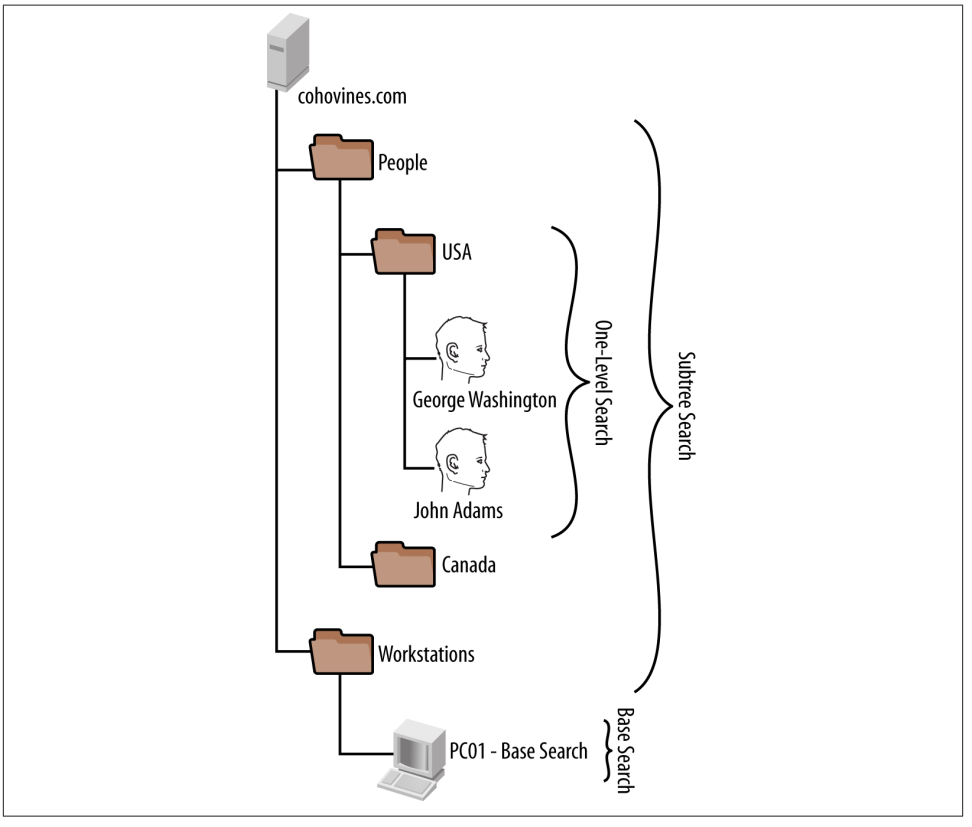


Figure 7-4. Search bases

Subtree searches look for every record under the specified search base. If you specify a one-level search, only records directly under a specific search base (e.g., an organizational unit), including possibly the search base itself, will be returned. Finally, base-level searches return only the object specified.

If you take a minute to refer back to Figure 7-1, you can see how the PDNT (parent DNT) and Ancestors columns make these searches efficient, since Active Directory can filter based on those values.

To specify a search base with AdFind, use the *-b* parameter. If you wanted to conduct a subtree search (the default) for users in the People OU, you would use this syntax:

```
adfind -f "(objectClass=user)" -b "OU=People,DC=cohovines,DC=com"
```

If you wanted to perform a one-level search for only users in the root of the USA OU, you would use syntax similar to the following (all on one line):

```
adfind -f "(objectClass=user)" -b "OU=USA,OU=People,DC=cohovines,DC=com"
  -s onelevel
```

Modifying Behavior with LDAP Controls

In addition to simply submitting an LDAP query, you may want to modify how the results are returned by the domain controller. You typically perform these modifications through the use of *LDAP controls*. LDAP controls are session options that are sent as part of the query via the LDAP protocol. The controls in a request are represented in the form of unique object identifier (OID) strings. You can use LDAP controls for common tasks such as sorting a result set on the server, paging the result set, and so forth.

Chances are that you aren't actually aware that an LDAP control has been loaded when making queries most of the time. For example, AdFind generally includes the LDAP control that enables paged result sets. The following list shows many of the LDAP controls that Active Directory supports with relation to searching, as well as how to use the more common controls with AdFind:

`LDAP_PAGED_RESULT_OID_STRING` *(1.2.840.113556.1.4.319)*
: Specifying this LDAP control instructs the domain controller that it can return more results than can fit in a single page. This is useful when searching for large result sets and should generally always be included in an Active Directory search.

`LDAP_SERVER_DIRSYNC_OID` *(1.2.840.113556.1.4.841)*
: DirSync is an LDAP feature that allows you to ask Active Directory for all the objects in a given naming context that have changed since the last time the search was performed. Changes are tracked with a cookie that is returned by the server.

  The DirSync control will only return modified attributes. If you want to return the full object when any attribute of that object has been modified, and you are running

Windows Server 2012 or later, use the `LDAP_SERVER_DIRSYNC_EX_OID` (1.2.840.113556.1.4.2090) version of this control instead.

Applications such as Microsoft Forefront Identity Manager (FIM) use the DirSync LDAP feature to track changes. For more information on this feature, refer to this link.

`LDAP_SERVER_DOMAIN_SCOPE_OID` *(1.2.840.113556.1.4.1339)*
: To make sure that the domain controller does not return referrals to result sets that are stored on other servers, include this LDAP control in your request.

`LDAP_SERVER_EXTENDED_DN_OID` *(1.2.840.113556.1.4.529)*
: When this LDAP control is enabled, Active Directory will return the SID and `objectGUID` of each result as prefixes to the object's DN in the result set. For more information, refer to this link.

`LDAP_SERVER_GET_STATS_OID` *(1.2.840.113556.1.4.970)*
: This extremely useful LDAP control instructs Active Directory to return statistics about how the query processor will perform the requested search, as well as performance statistics about the result. We'll discuss this feature in more detail later in this chapter.

`LDAP_SERVER_NOTIFICATION_OID` *(1.2.840.113556.1.4.528)*
: When this control is specified, Active Directory won't return a result until the requested object is modified. This is useful for tracking changes to specific objects in the directory and responding to them. For more information, refer to this link.

`LDAP_SERVER_RANGE_OPTION_OID` *(1.2.840.113556.1.4.802)*
: Similar to the paged results control mentioned earlier, this is typically a control you will always want to specify. The ranged results control is used when you need to retrieve values in a multivalued attribute in excess of the maximum number of values a DC will return by default. You can use the `LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID` (1.2.840.113556.1.4.1948) alternate implementation of this LDAP control to ensure that errors will not be returned if you request more values than are available.

`LDAP_SERVER_SD_FLAGS_OID` *(1.2.840.113556.1.4.801)*
: Use this control to tell the domain controller which components of an object's `ntSecurityDescriptor` (the ACL) to retrieve. Depending on the permissions of the user performing the query, not all of the components of the ACL may be readable. For more information, refer to this link.

`LDAP_SERVER_SEARCH_OPTIONS_OID` *(1.2.840.113556.1.4.1340)*
: This generic control's most useful function is called *phantom root*. The phantom root feature enables you to perform a search across all of the naming contexts (application NCs excluded) hosted on a global catalog. If you have a multidomain

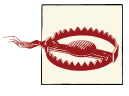
forest with disjointed namespaces, use this control to search across all of the domains at once. Use the *-pr* switch to enable this function in AdFind.

`LDAP_SERVER_SHOW_DELETED_OID` *(1.2.840.113556.1.4.417)*
: This control instructs the domain controller to include deleted objects and tombstones in the result set. If you have enabled the Active Directory Recycle Bin, this will only include objects that are currently recoverable. To include objects that have transitioned out of the Recycle Bin, you must also include the `LDAP_SERVER_SHOW_RECYCLED_OID` (1.2.840.113556.1.4.2064) control.

  Active Directory treats deactivated linked values (links to objects that are in the Recycle Bin) differently. If you want to include deactivated links in your results, add the `LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID` (1.2.840.113556.1.4.2065) control.

  To include deleted objects in AdFind, append the *-showdel* switch. If you also want to include objects that have transitioned out of the Recycle Bin, also append the -showrecycled switch. To include deactivated links, append *-showdelobjlinks*.

`LDAP_SERVER_SORT_OID` *(1.2.840.113556.1.4.473)*
: This control instructs the server to sort the results based on one attribute before returning the result set. You can request search results to be sorted with AdFind by using the *-sort* and *-rsort* (reverse order) parameters. This document provides a great deal more information about sorting, especially with regard to the language-specific ordering of a result set and phonetic sort functionality on Japanese-language domain controllers.



Server-side sorting requires the use of a temporary table in the Active Directory database when the attribute being sorted on is not indexed. Temporary tables are limited in size. Consequently, server-side sorting of large result sets with unindexed attributes will likely fail due to the size constraints of the temporary table.

`LDAP_CONTROL_VLVREQUEST` *(2.16.840.1.113730.3.4.9)*
: Virtual list view (VLV) searches are useful for large searches that will be paged through in a format similar to scrolling through an address book or phone directory. In fact, some versions of Microsoft Exchange use VLV for building the address books shown in email clients. For more information, refer to this link.

`LDAP_SERVER_ASQ_OID` *(1.2.840.113556.1.4.1504)*
: Attribute scoped queries (ASQs) are useful when you want to perform a query based on a linked attribute's value(s). For example, you might want to return all of the users who are members of a group called *All Users*. You can do this using AdFind with this syntax:

```
adfind -asq member -b "cn=All Users,ou=Groups,dc=cohovines,dc=com"
 -f "objectClass=user"
```

In the case of LDAP controls that we didn't demonstrate with AdFind in the preceding list, you can use LDP to exercise many controls. In most cases, go to Browse→Search and then click Options. On the Options screen, click Controls, as shown in Figure 7-5. Use the "check in" button to add a control to the request once you configure it on the lefthand side of the screen. Virtual list view (VLV) searches can be conducted by using the dedicated VLV option available under Browse→Virtual List View.

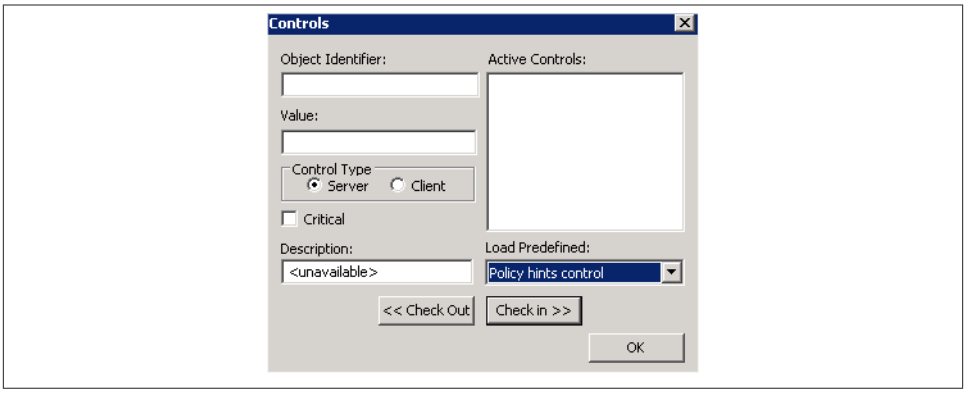


Figure 7-5. LDP Controls dialog

Attribute Data Types

Certain types of attributes are more challenging to work with than what we have looked at so far. These types of attributes include dates and times (such as the `pwdLastSet` attribute) and bit masks (such as the `userAccountControl` attribute). Active Directory supports querying these attributes; it just takes a bit of additional effort.

Dates and Times

When you need to find users with expired passwords or objects that were created before a certain time, you'll need to work with the formats that Active Directory stores times in. These formats vary by attribute. The two most common are the 64-bit NT `FILETIME` format and the more standards-based generalized time syntax.

NT `FILETIME`s are integers that count the number of 100-nanosecond intervals since January 1, 1601. Generalized time, on the other hand, simply stores a timestamp in a more legible format: *`YYMMDDHHMMSS[.fffZ]`*. This format is year, month, day, hour, minute, and second. The optional three-digit decimal value is fractions of a second. The trailing `Z` indicates that the time is universal or *Zulu* time. If the `Z` is not present, the time is local time.

Encoding values for filters can be accomplished in a couple of ways. In the case of an attribute such as `pwdLastSet`, which is stored in `FILETIME` format, you can ask AdFind to perform the conversion for you. To retrieve all of the computers whose passwords were last changed before October 1, 2012, for example, you would use this AdFind command:

```
adfind -f "(&(objectClass=computer)(pwdLastSet<={{LOCAL:2012/10/01-12:00:00}}))"
 -binenc
```

If you are using a tool other than AdFind to perform your search, you'll need to convert the timestamp in advance. You can use Windows PowerShell to perform the conversion: *[DateTime]::Parse("10/01/2012").ToFileTime()*. Once you get the result (in this case, `129935412000000000`), construct the filter as follows:

```
(&(objectClass=computer)(pwdLastSet<=129935412000000000))
```

Searches for generalized time attributes such as `whenCreated` and `whenChanged` should be formatted using the syntax described earlier (*`YYMMDDHHMMSS.fffZ`*).

When results are returned, they will be in the same format in which Active Directory expects values to be provided. With AdFind, you can use the *-tdcs* switch to decode `FILETIME` fields (such as `pwdLastSet`) and the *-tdcgts* switch to decode generalized time fields (such as `whenChanged` and `whenCreated`).

Much like converting a value to `FILETIME` format with PowerShell, you can convert a `FILETIME` back to readable format by running the following: *[DateTime]::FromFileTime(129935412000000000)*.

Bit Masks

The most common example of using a bit mask in a search filter will undoubtedly be to find users that are enabled or disabled. This flag is tracked as the second bit in the `userAccountControl` attribute. Active Directory supports two special matching rules in the LDAP filter to perform AND and OR operations in a filter.

For example, to find users who are disabled, use this filter:

```
(&
    (objectCategory=person)
    (objectClass=user)
    (userAccountControl:1.2.840.113556.1.4.803:=2)
)
```

The opposite, finding users who are enabled, requires the use of the NOT operator:

```
(&
    (objectCategory=person)
    (objectClass=user)
    (!
          (userAccountControl:1.2.840.113556.1.4.803:=2)
    )
)
```

The OR operator is accessible by specifying the `1.2.840.113556.1.4.804` OID in the filter.

If you're using AdFind, you can simplify these searches quite a bit through the use of the *-bit* parameter. More specifically, you can execute the following command to find all of the users who are disabled:

```
adfind -f
 "(&(objectCategory=person)(objectClass=user)(userAccountControl:AND:=2))" -bit
```

The In-Chain Matching Rule

One last LDAP filter trick that's worth mentioning is the *in-chain matching rule*. This modifier allows you to ask Active Directory to walk a group's membership chain forward or backward, to either determine all the groups a user is a member of, or determine whether or not a user is indirectly a member of a group (e.g., via group nesting).

You can use the in-chain matching rule much like the AND and OR operator matching rules discussed previously. For example, to find out if a user is a member of the group *Important People*, you would perform a base search of the user on the `memberOf` attribute. You can do this with AdFind using syntax similar to the following (all on one line):

```
adfind -s base -b "cn=Brian,OU=People,DC=cohovines,DC=com" `
  -f (memberof:INCHAIN:=(cn=Important People,OU=Groups,DC=cohovines,DC=com)) `
  -bit
```

You can also do what is effectively the opposite and return all of the groups that user *Brian* is directly or indirectly a member of, using a similar search:

```
adfind -f (member:INCHAIN:=(cn=Brian,OU=People,DC=cohovines,DC=com)) -bit
```

The `INCHAIN` shortcut in AdFind simply substitutes the actual matching rule OID: `1.2.840.113556.1.4.1941`.

Optimizing Searches

Thus far we've looked at the multitude of ways we can ask Active Directory to return data via LDAP. One very important component of this task that we haven't explored yet is performance. Submitting a search to a domain controller that brings the server to its knees is obviously counterproductive. Still, experienced Active Directory administrators will inevitably be able to recount more than one event where a domain controller was brought down by an application that created so much load through an inefficient query that the CPU resources of the server were completely consumed.

Efficient Searching

At a minimum, efficient searching comes down to knowing whether or not indexes will be used for your filter, as well as the mere practicality of the server being able to rapidly find the results that meet your request. *Indexes* are the key component of an efficient search. An index is a database-level function that enables the server to quickly find results matching a value. If an index isn't available, the server must scan the entire database to find rows matching your request.

In Chapter 5, we talked quite a bit about how to index an attribute as well as the different types of indexes available. For simplicity, we've reproduced part of one of the tables from Chapter 5 in Table 7-3, which details the different types of indexes you can add to an attribute.

Table 7-3. searchFlags bits

Bit number	Value	Description
1	1 (0x0001)	Create an index for the attribute. All other index-based flags require this flag to be enabled as well. Marking linked attributes to be indexed has no effect.
2	2 (0x0002)	Create an index for the attribute in each container. This is only useful for one-level LDAP queries.
6	32 (0x0020)	Create a tuple index. Tuple indexing is useful for medial searches. A medial search has a wildcard at the beginning or in the middle of the search string. For example, the medial search (`drink=*coke`) would match Cherry Coke, Diet Coke, etc.
7	64 (0x0040)	Create a subtree index. This index is designed to increase the performance of VLV queries.

Generally speaking, the most common type of index you're going to add is the standard index listed first in Table 7-3. If you're going to be performing medial searches, you should consider adding a tuple index (bit 6). A medial search such as (`title=*vice*`) finds any object with a job title containing the term *vice* (such as vice president).

Using the stats control

The `stats` LDAP control is available if you are a domain administrator or you have debugging privileges on the targeted domain controller. The best way to access the `stats` control is with AdFind and the *-stats+* switch.

Make sure you run AdFind from an elevated command prompt in order to use the `stats` control.

The `stats` control will give you a number of data points in the output, including which indices were used to process the search, the number of records that had to be individually scanned (versus the number of records that were returned), as well as timing information. Consider the following output for a search of computers running Windows XP:

```
Statistics
================================
Elapsed Time: 156 (ms)
Returned 1267 entries of 3630 visited - (34.90%)

Used Filter:
 ( & (objectClass=computer) (operatingSystem=*XP*) )

Used Indices:
 idx_objectClass:3798:N

Pages Referenced          : 83092
Pages Read From Disk      : 1
Pages Pre-read From Disk  : 0
Pages Dirtied             : 0
Pages Re-Dirtied          : 21
Log Records Generated     : 0
Log Record Bytes Generated: 0


Analysis
--------------------------------
Hit Rate of 34.90% is Ok

Indices used:

Index Name  : idx_objectClass
Record Count: 3798  (estimate)
Index Type  : Normal Attribute Index


Filter Breakdown:

(
 (&
   (objectClass=computer)
   (operatingSystem=*XP*)
 )
)
```

From this listing, we can learn quite a bit about the search query that was submitted:

Elapsed Time
: How long did it take the domain controller to process the search?

Returned X entries of Y visited
: This tells us how many records the domain controller had to scan through versus the number of results that were actually returned. A low percentage here tells us that there is not good index coverage for our query.

Used Indices
: What attribute indices were used to filter the result set? In this case, the domain controller filtered the results based on the `objectClass` index to all of the computer objects in the domain. Once this index was used, the DC had to scan each computer to check if its `operatingSystem` attribute matched the search filter.

Pages Referenced, etc.
: These statistics tell us how many database pages (4 KB units of storage) were examined. The Pages Read From Disk indicator tells us how much data was cached in RAM. In this case, virtually all of the data was cached in RAM, which greatly improved the speed of the search.

Hopefully you can see from this example how valuable the `stats` control is when tuning a query. The data returned tells you what percentage of the data was able to be located solely via indices versus how many records had to be evaluated individually. The pages referenced and pages read from disk statistics give an excellent idea of performance with relation to hardware. In this case, virtually the entire result set was already cached in memory, leading to much better performance.

In the case of this sample, the domain is relatively small, with only a few thousand computers. In a large domain, this query would take much longer to process and the addition of a tuple index on the `operatingSystem` attribute would likely improve performance.

Consider taking a few minutes to run through the sample queries in this chapter with the `stats` control enabled. Look at the results and think about how the components of the query might have affected Active Directory's processing choices. Refer back to the discussion of the Active Directory database structure earlier in this chapter as you think about this.

objectClass Versus objectCategory

One of the urban legends dating back to the early days of Active Directory revolves around why the `objectClass` attribute wasn't indexed by default until Windows Server 2008. Originally it was thought that indexing multivalued attributes (such as `objectClass`) would lead to poor performance. As a result, Windows 2000 shipped with an unindexed `objectClass` attribute and an additional single-valued attribute called `objectCategory` that is indexed.

This decision also led to many performance issues related to queries that only used the `objectClass` attribute, and recommendations abound that queries always use the `objectClass` *and* `objectCategory` attributes in order to ensure the best performance. If you want to be sure that your query will perform well on all versions of Active Directory, we recommend that you use `objectClass` and `objectCategory`. If you have indexed `objectClass` in your directory, or all your domain controllers are running Windows Server 2008 or better, then you can feel free to use `objectClass` independently without fear of performance repercussions.

Summary

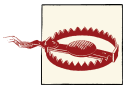
In this chapter, we looked at the query language that retrieves data from Active Directory: LDAP. First we looked at how the database is organized, to give you a foundation for understanding how Active Directory retrieves the data; then we jumped into the LDAP filter syntax and handling special data types. Finally, we looked at performance and how to ask domain controllers for performance information related to the queries you submit. All Active Directory administrators will need to create a report of directory data or integrate an application sooner or later in their careers, and we hope that this chapter helps you complete those tasks.

CHAPTER 8

Active Directory and DNS

One of the big advantages of Active Directory over its predecessor, Windows NT, is its 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 has become a candidate for elimination on many enterprise networks.

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 many dependencies on the DNS infrastructure. This is only potentially because it depends 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.



Although Active Directory doesn't need WINS, or more accurately, NetBIOS name resolution, other systems and technologies may require it. Many administrators are quick to try to remove WINS from their environment, but generally speaking, the administrative cost of maintaining a WINS infrastructure is substantially smaller than the cost involved in executing a project to remove WINS.

The intent of this chapter is to provide you with a good understanding of how Active Directory uses DNS and to review some of the options for setting it up within your organization. We will briefly touch on some DNS basics, but we 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 2003* by Matt Larson, Cricket Liu, and Robbie Allen (O'Reilly).

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 following RFCs 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.
- Dynamic DNS allows clients to add and delete resource records dynamically.

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* namespace was delegated to the name server *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, *mycorp.com* could delegate *subdomain1.mycorp.com* to the name server *ns2.mycorp.com*. At that point, *subdomain1.mycorp.com* becomes its own zone, for which *ns2.mycorp.com* is authoritative.

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 8-1 lists some of the more common resource record types.

Table 8-1. Commonly used resource record types

Record type	Name	Description
A	Address record	Maps a hostname to an IPv4 address.
AAAA	Address record	Maps a hostname to an IPv6 address.
CNAME	Alias record	Maps an alias to a hostname.
MX	Mail Exchanger record	Specifies a mail route for a domain.
NS	name server record	Specifies a name server for a given domain.
PTR	Pointer record	Maps an IPv4 address to a hostname.
SOA	Start of Authority record	Contains administrative data about a zone, including the primary name server.
SRV	Service record	Maps a particular service (e.g., LDAP) to one or more hostnames and ports.
TXT	Text record	Contains arbitrary text data. Often used for protocols that take advantage of DNS without a specific DNS record type. Sender Policy Framework (SPF) for spam filtering is one example.

One important resource record to note is the SRV record type. SRV records are used extensively by domain controllers and Active Directory clients to locate servers that have a particular service. We will describe how Active Directory uses these records in more detail later in the chapter.

It is also important to note that SRV records are dependent upon A records. Each SRV record provides a variety of details about each service it represents (LDAP, for example), including the computers on which the services run. SRV records achieve this by maintaining pointers to the fully qualified names of the A records that describe the computers running the services.

Client Lookup Process

The process for a client to resolve a DNS name is relatively straightforward, but depending on the name being resolved, the number of hops involved can vary dramatically. Figure 8-1 shows the hops involved for a machine to resolve the name *www.whitehouse.gov* using a caching DNS server (such as from the client's Internet service provider, or ISP).

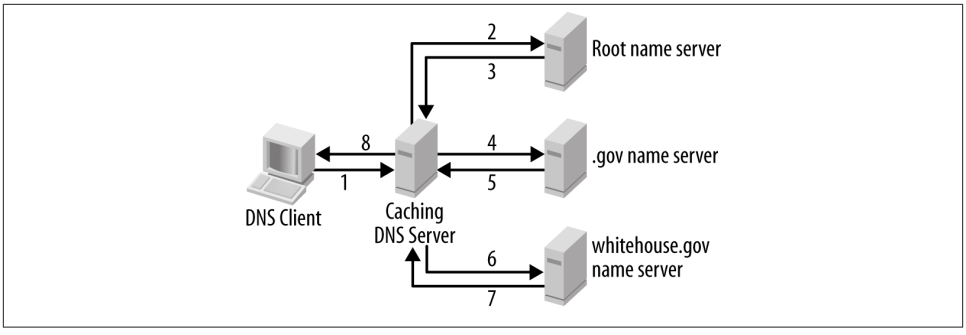


Figure 8-1. Client lookup process

The lookup process shown in Figure 8-1 involves eight steps:

1. The client sends a request to his DNS server for *www.whitehouse.gov*.
2. The client's DNS server makes a request to one of the root name servers for assistance resolving the name.
3. The root name server refers the client to the name servers for the *.gov* domain.
4. The client's DNS server makes a request to the *.gov* name server for assistance resolving the name.
5. The *.gov* name server refers the client to the name servers for the *whitehouse.gov* domain.
6. The client's DNS server makes a request to the *whitehouse.gov* name server for *www.whitehouse.gov*.
7. The *whitehouse.gov* name server returns the A record for *www.whitehouse.gov*.
8. The client's DNS server returns the response to the client.

As you can see, there are a number of hops involved. The name servers that clients use to resolve Internet names are often referred to as *caching DNS servers* because they cache a copy of the responses they get from other DNS servers for a period of time, to make future lookups faster. The linkage between the root name server and the *.gov* name server and the *.gov* name server and the *whitehouse.gov* name server is called a delegation. We'll discuss delegations more later in this chapter. In a nutshell, the delegation contains the name server (NS) records that are necessary to point the client to the name servers responsible for the delegated domain.

Dynamic DNS

Dynamic DNS (DDNS), defined in RFC 2136, is a method for clients to send a DNS server request to add or delete resource records in a zone. Having this capability has greatly increased the supportability of DNS in large environments. Before DDNS, the

primary means to update a zone was by either directly editing a text-based zone file or via a vendor-supported GUI, such as the Windows DNS MMC snap-in.

Active Directory takes full advantage of DDNS to ease the burden of maintaining the resource records it requires. Each domain controller can have anywhere from a few dozen to a few hundred associated resource records, depending on the size of the Active Directory site topology. Any time the site topology changes, one or more domain controllers may need to change some or all of the resource records previously registered. Because of this dynamic nature of Active Directory resource records, in a large environment it could easily require a full-time employee to manually maintain all of the records. For more information about the security of dynamic DNS, see the sidebar "Securing Your Dynamic Updates" on page 173.

Securing Your Dynamic Updates

The RFC that defined Dynamic DNS, RFC 2136, did not provide for a security model to secure updates from clients. As you might expect, this is a very serious limitation to wide-scale adoption. To address this problem, RFC 2137, *Securing Your Dynamic Updates* was created. Unfortunately, RFC 2137 was not very practical in implementation and tended to be overly complex. Later, RFC 2535, "Domain Name System Security Extensions," defined a public key-based method for securing DNS requests, commonly known as *DNSSEC*. RFC 3007 was then created, which rendered RFC 2137 obsolete and updated RFC 2535 to provide a more flexible method to secure update requests. Many DNS server products have only recently started to provide support for these RFCs, and only time will tell whether they will become widely adopted. Check out the following for more information on RFCs 2535 and 3007:

- *http://www.ietf.org/rfc/rfc2535.txt*
- *http://www.ietf.org/rfc/rfc3007.txt*

Although Windows Server 2008 provides support for some of the resource record types defined in RFC 2535, such as KEY, SIG, and NXT, it does not provide full compliance, such as message signing and verification.

Microsoft uses access control lists (ACLs) in Active Directory to provide secure dynamic updates. Zones that are Active Directory-Integrated (described later in the chapter) store their DNS data in Active Directory. You can then set ACLs on the DNS-related objects in Active Directory to grant or deny security principals permission to add and/or update records. By default, authenticated computers in a forest can make new entries in a zone, but only the computer that created an entry is allowed to modify the data associated with that entry.

RFC 2136 can be found at *http://www.ietf.org/rfc/rfc2136.txt*.

Global Names Zones

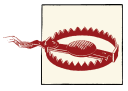
With each version of Windows since Windows 2000, Microsoft has continued to invest in making it more practical for some organizations to retire their WINS infrastructures. WINS is very useful in some organizations because it supports short name resolution—in other words, unlike with DNS, you do not need to specify a hierarchical name to resolve.

You can mitigate this with DNS by configuring DNS-suffix search orders on clients: the DNS resolver on the client will attempt to resolve the short name by appending each DNS suffix, defined one at a time in the order listed. In a large organization with numerous DNS namespaces, this list of suffixes could potentially be quite long and difficult to maintain, and would also cause significant increases in network traffic when trying to resolve short hostnames.

Windows Server 2008 introduced new DNS server functionality called the *global names zone*, which you can configure to support short name resolution via DNS without the DNS-suffix search-list requirements on your clients. Any client that supports DNS resolution can utilize the global names zone functionality without additional configuration. Windows Server 2008 and newer DNS servers will first try to resolve the name queried in the local zone, and if that fails, they will then try to resolve it in the global names zone.

Global names zones may be particularly useful to you if you are looking to deploy IPv6 in your organization, as WINS does not support IPv6 addressing. When evaluating whether or not to deploy a global names zone, consider the following:

- Are you retiring WINS, or planning to?
- Are all of your DNS servers that clients contact running Windows Server 2008 or newer? Windows Server 2003 and earlier DNS servers do not implement the global names zone functionality.
- Are you able to manage the static registration of global names? Dynamic registration of records in the global names zones is *not* supported, so you will need to manage the registrations manually or with a script.
- Do you need the functionality global names zone offers? This feature can potentially increase management overhead, so you should consider whether you will benefit from it before investing the time in deploying it.



While Windows Server 2008 does not support dynamic registration of records in the global names zone, it does check for uniqueness when clients register new records in other zones. If a client tries to register a new record for a hostname that is already defined in the global names zone, this registration will fail.

Deploying a global names zone is a three-step process:

1. Create a forward lookup zone on a DNS server called *GlobalNames*.
2. Enable global names zone support on every DNS server that will host the *GlobalNames* zone:

```
dnscmd ServerName /config /EnableGlobalNamesSupport 1
```

3. Configure replication of the *GlobalNames* zone. It is recommended that you use the *ForestDnsZones* application partition for this step.

If you have multiple forests and you want to maintain a single *GlobalNames* zone, you will need to configure an SRV record in the *_msdcs.<other forest>* zone in the other forests. This SRV record is called *_globalnames*, and it should point to the FQDN of a server hosting the actual *GlobalNames* zone.

You will need to perform step 2 on all of the DNS servers in the other forest in order for them to honor the `_globalnames` SRV record.

Once you have the zone created, you will need to populate it with records so that name resolution can occur. The recommended approach is to place CNAME records in the *GlobalNames* zone and alias them to the records for the specific server/name in the relevant forward lookup zone.

DNSSEC

One of the challenges with DNS is that there is no mechanism in the base protocol to verify that responses are authentic and accurate. In between the client machine and the authoritative DNS server hosting a given zone, there are often one or more nonauthoritative caching DNS servers involved. For example, when you browse to *www.whitehouse.gov* on your home computer, chances are your DNS query will go to your ISP's caching DNS servers. Your ISP's DNS server will in turn recursively resolve the query (*www.whitehouse.gov*) and then return the response to you.

The problem that DNSSEC solves is what happens if your ISP's DNS server cache is poisoned by an attacker. If the cache is poisoned, the IP address returned might not be

the actual IP address. With DNS, there is no way for the client to have a level of assurance about the validity of the response returned. DNSSEC provides the extensions to DNS to ensure that DNS responses are authentic.

For an in-depth compilation of DNSSEC resources, check out *http://www.dnssec.net*.

How Does DNSSEC Work?

DNSSEC depends on two key components. The first component is a trust chain that begins with the root DNS servers, or with *trust anchors* that are deployed in your internal DNS infrastructure. The root DNS name servers are globally distributed and are responsible for the first hop when you want to resolve a name on the Internet. The second component is the publication of digital signatures in DNS for individual records and the zone as a whole. Combined, these two components enable you to verify the authenticity of a given DNS response.

Verifying the authenticity of a DNS response can happen in two places. The first place this can happen is on the DNS server that your client queries to resolve a name. The second possible location is in the DNS client on the machine making the query. The Windows DNS client (beginning with Windows 7 and Windows Server 2008 R2) is DNSSEC-aware but nonvalidating. Effectively, this means that the Windows DNS client is dependent on the upstream DNS servers to support DNSSEC. If you're thinking about DNSSEC for resolving names on the Internet, your ISP will need to support DNSSEC on its DNS servers, and the zones hosting the records you want to resolve will need to be signed. If you're thinking about internal DNS, you can deploy Windows Server 2012 DNS servers and/or Windows Server 2012 domain controllers to take advantage of DNSSEC.

Resource records

DNSSEC uses a number of new resource record types. These record types are listed in Table 8-2.

Table 8-2. DNSSEC record types

Record Type	Description
RRSIG	Stores a digital signature for a single resource record
DNSKEY	Publishes a public key used when signing DNS data
DS	Identifies the signing key used in a delegated DNS zone
NSEC3	The next Secure response record, used for providing an authenticated response that a name is not resolvable (a negative response) in the zone

The DNSKEY record stores two types of public keys: the key signing key (KSK) and the zone signing key (ZSK). The KSK is used only to sign other keys—namely, ZSKs. ZSKs are in turn used to sign records in the zone. Over time, it is important to be able to change the keys used to sign records. Windows Server 2012 provides automatic key management, but in the case of the KSK, you will have to work with the owner of the parent zone (e.g., *whitehouse.gov* would need to work with the owner of *.gov*) to update the Delegated Signer (DS) record when the KSK changes. The DS record contains a hash of the KSK and is used to verify the authenticity of a response.

Lookup process

If you refer back to Figure 8-1 and the accompanying text, you'll see that there are a number of hops involved in resolving a name such as *www.whitehouse.gov*. We've specifically chosen *www.whitehouse.gov* as an example in this chapter because the *whitehouse.gov* DNS zone is signed and supports DNSSEC. In fact, in Step 7 of Figure 8-1, when the *whitehouse.gov* name server returns the *www* A record, the digital signature is also returned in an RRSIG record.

The steps necessary to verify the validity of the *www.whitehouse.gov* A record are:

1. The caching DNS server requests the zone signing key DNSKEY record and the corresponding RRSIG record for the *whitehouse.gov* zone.
2. The *whitehouse.gov* name server returns the DNSKEY and RRSIG records.
3. The caching DNS server computes the hash of the *www.whitehouse.gov* A record.
4. The caching DNS server decrypts the RRSIG record for *www.whitehouse.gov* with the *whitehouse.gov* ZSK to get the RRSIG record hash.
5. The caching DNS server compares the hashes and ensures they match.

At this point, assuming everything checks out, we have validated the response received, but we have no way of guaranteeing its authenticity. If this were the end of the validation process, there would be no way of guaranteeing that the DNS server's cache wasn't also poisoned with invalid RRSIG and DNSKEY records. So, we continue:

6. In order to verify the authenticity of the response, the caching DNS server requests the key signing key for the *whitehouse.gov* zone.
7. Next, the *whitehouse.gov* ZSK is validated by comparing the hash of the ZSK and the ZSK's RRSIG record.

Unlike before, the RRSIG for the zone's zone signing key is encrypted with the zone's key signing key.

To fully authenticate the response, we need to walk the trust chain. The trust chain exists in the reverse path of the DNS resolution that took place in Figure 8-1:

8. The caching DNS server asks the *.gov* name server for the DS record for *whitehouse.gov*.
9. The *.gov* name server returns the DS record and the corresponding RRSIG record.
10. To validate the response from *whitehouse.gov*, the server compares the hash of the *whitehouse.gov* KSK with the data in the DS record.

This validation is reliable because the DS record in the *.gov* name server was provided by the administrator of the *whitehouse.gov* name server. Later, when we configure DNSSEC, you'll see where you get a copy of the data necessary for the DS record. Unfortunately, we're not done yet. We now need to validate the authenticity of the data returned from the *.gov* name server:

11. The caching DNS server asks the root name server for the DS (delegated signer) record for *.gov*.
12. The root name server returns the DS record and the corresponding RRSIG record.
13. To validate the response from *.gov*, the server compares the hash of the *.gov* KSK with the data in the DS record returned from the root name server.

Finally, you may be wondering how we validate the response from the root name server. Ultimately, we need to have a well-known trust point. If you're familiar with public key infrastructure (PKI) and certificates, this is much like the trusted root store in Windows. DNS stores trust anchors that contain the necessary data to validate the response from the root name servers and other name servers that are not related to a trusted set of name servers.

If you want to try making a query with DNSSEC enabled, you can do so with the *Resolve-DnsName* PowerShell cmdlet that was added in Windows 8 and Windows Server 2012. For example, to resolve *www.whitehouse.gov*, you would run `Resolve-DnsName www.whitehouse.gov -DnssecOk`. You will get a response back similar to the following. Note the signature information contained in the response:

```
Name                           Type   TTL   Section    NameHost
----                           ----   ---   -------    --------
www.whitehouse.gov             CNAME  0     Answer     www.whitehouse.gov.ed…
www.whitehouse.gov.edgesuite.net CNAME  0     Answer     www.eop-edge-lb.akadns…
www.eop-edge-lb.akadns.net     CNAME  0     Answer     a1128.h.akamai.net

Name        : www.whitehouse.gov
QueryType   : RRSIG
TTL         : 2348
Section     : Answer
TypeCovered : CNAME
```

```
Algorithm   : 7
LabelCount  : 3
OriginalTtl : 3600
Expiration  : 8/15/2012 10:23:13 PM
Signed      : 8/12/2012 9:23:13 PM
Signer      : whitehouse.gov
Signature   : {34, 111, 6, 55...}


Name       : a1128.h.akamai.net
QueryType  : A
TTL        : 18
Section    : Answer
IP4Address : 23.62.63.153


Name       : a1128.h.akamai.net
QueryType  : A
TTL        : 18
Section    : Answer
IP4Address : 23.62.63.152


Name      : .
QueryType : OPT
TTL       : 32768
Section   : Additional
Data      : {}
Enforcing DNSSEC Validation
```

As we discussed earlier, the DNS client in Windows does not perform validation of responses locally. Instead, the client depends on its DNS server to perform the validation for it. There are one or two steps involved in configuring this. First, on the properties of the clients' DNS servers, you must ensure that "Enable DNSSEC validation for remote responses" is checked, as shown in Figure 8-16 in the section "Aging and Scavenging" on page 201. Next, if you want to ensure that clients are receiving validated responses for some (or all) domains, you will need to configure the client's Name Resolution Policy Table (NRPT).

The easiest way to configure the NRPT is with Group Policy. You can access the NRPT by browsing to *Computer Configuration\Policies\Windows Settings\Name Resolution Policy*. Figure 8-2 shows the NRPT with a rule requiring that DNS requests for *whitehouse.gov* have been validated.

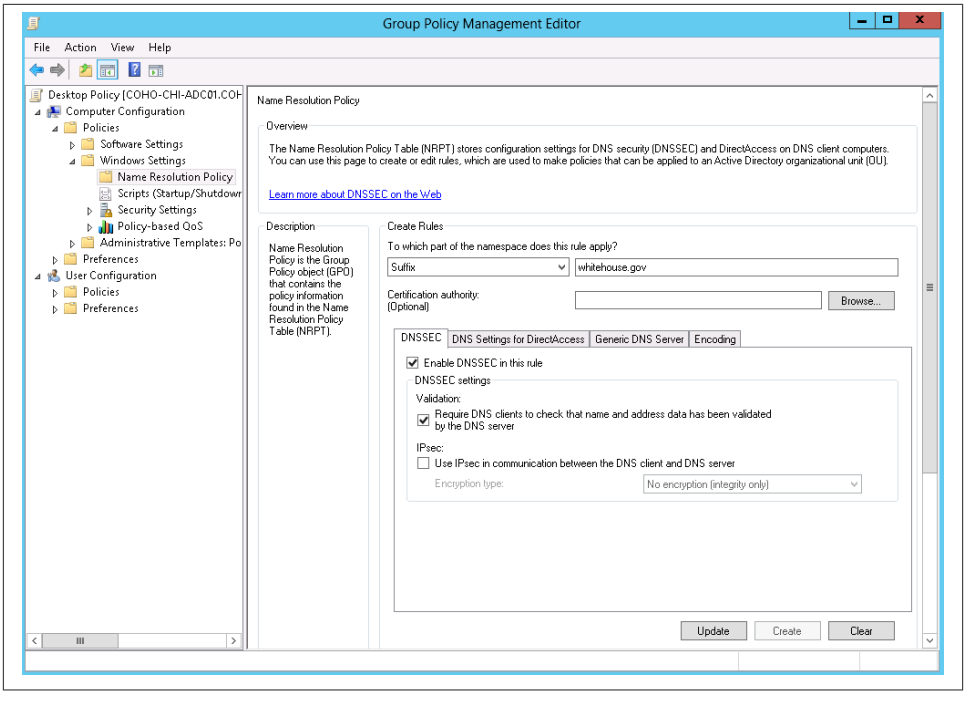


Figure 8-2. Name Resolution Policy Table configuration with Group Policy

The downside to Windows not being able to perform validation in the local client is that there is a vulnerability in the last hop between the client and the client's DNS server. It is possible that the response could be tampered with during that last hop. Thus, to truly have complete end-to-end DNS validation, you will need to secure the last hop. One possible solution to this problem is to deploy IPSEC and use it to authenticate the connection between the client and the DNS server.

Configuring DNSSEC for Active Directory DNS

Up until now, we have discussed DNSSEC in the context of resolving a name on the Internet. While the principles we discussed are identical, the implementation is slightly different for Active Directory DNS. Namely, we'll be signing dynamic zones and we'll deploy trust anchors to Active Directory rather than relying on the recursive process of contacting parent name servers described earlier.

Before you can complete your DNSSEC deployment, you will need to upgrade all of your DNS servers to Windows Server 2012. Windows Server 2008 R2 DNS servers support DNSSEC validation, but they don't support Active Directory-integrated trust anchors or signing of dynamic DNS zones.

The RRSIG records are not stored in Active Directory, as they would create a substantial amount of additional Active Directory replication traffic. Instead, the records are computed at server startup (and during record update/registration) and then stored in memory. When the DNS server service is started, all of the records in the zone will need to be signed to create RRSIG records. In a large zone this could take several minutes. Depending on the hashing algorithm and key length you choose, the speed will vary greatly. Both of these factors will also affect the amount of memory used by the DNS server service to store the RRSIG records in memory.

To begin the DNSSEC signing process, right-click the zone you want to sign and click DNSSEC→Sign the Zone. You'll first be asked whether you want to create custom signing parameters, duplicate the signing parameters of another zone, or use the defaults. We'll create custom signing parameters.

You'll need to select one DNS server as the key master for the zone, as shown in Figure 8-3. The key master is responsible for generating new signing keys when they are requested or when keys expire.

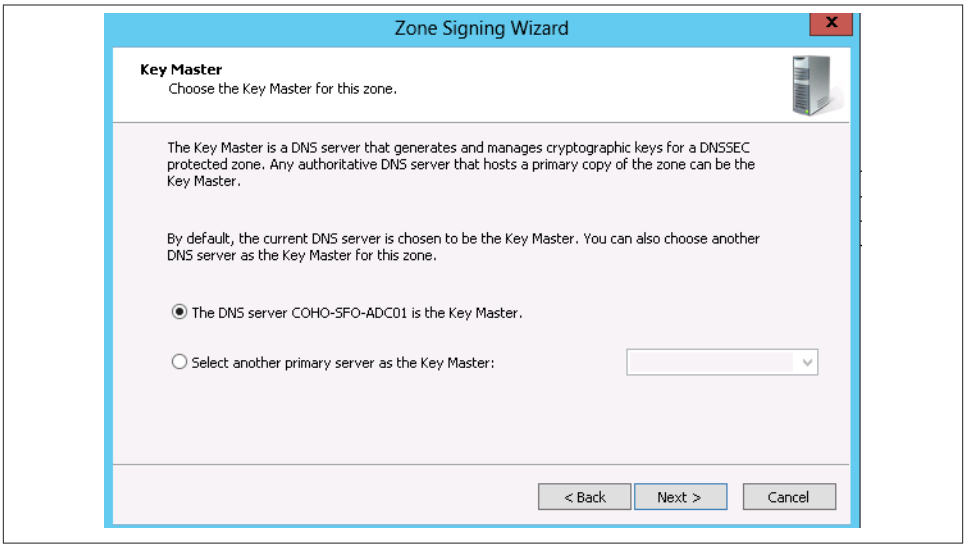


Figure 8-3. Nominating the DNSSEC key master for the zone

Next, you'll be prompted to generate a key signing key, as shown in Figure 8-4. KSKs are used to sign other types of signing keys. Generally speaking, you should be able to accept the defaults unless you have specific security requirements that mandate changing the defaults. Keep in mind that using a longer key or a stronger cryptographic algorithm will increase the CPU requirements and DNS server service startup times based on the increased mathematical workload.

The parameters shown in Figure 8-4 are:

Cryptographic algorithm
: This is the type of hash to compute for the signature. You can define multiple key signing keys with different algorithms if you need to support multiple algorithms.

Key length
: This is the number of bits that comprise the key. Choosing the key length is a balance between cryptographic strength and computational requirements to compute the hash.

Key storage provider
: This is where the key will be stored. For Active Directory DNS you must use the Microsoft Software Key Storage Provider. If you were signing a static standalone zone, you could use a hardware security module (HSM), for example, to protect the private key.

DNSKEY RRSET signature validity period
: This is time period (in hours) that a signature generated with this key is valid.

Key rollover
: Over time, you will want to change the key that is used for signing in order to ensure that it is not compromised by brute force. Windows Server 2012 will automatically publish a new key on a periodic basis if requested. You may need to coordinate the publication and updating of the Delegated Signer (DS) record if you roll over the KSK.

The next key you will need to configure is the zone signing key (ZSK), as shown in Figure 8-5. The zone signing key is used to compute digital signatures (RRSIG records) for DNS records in the zone. The parameters shown in Figure 8-5 are much the same as those in Figure 8-4. The exception is that the default recommended key length is shorter and automatic rollover is more frequent. The default key length is shorter because the ZSK is used much more often (any time a record is created or updated, or for each record on service startup). The shorter key length lessens the CPU impact. The shorter key length is compensated for with a more frequent rollover. The ZSK does not have to be published anywhere outside of the zone, so there is much less overhead involved in a ZSK rollover.

Next, you will be prompted to choose between NSEC and NSEC3 records, as shown in Figure 8-6. NSEC records are used to provide an authentic response that denies a record exists. More specifically, if you request a record (such as *some-record.contoso.com*) and there is no record called *some-record*, rather than returning an error (which could be spoofed), the server returns a signed response in the form of an NSEC or NSEC3 record. Generally speaking, we recommend the use of NSEC3 over NSEC. One of the shortcomings of NSEC records was that they could enable a dictionary attack that disclosed every single record in the DNS zone.

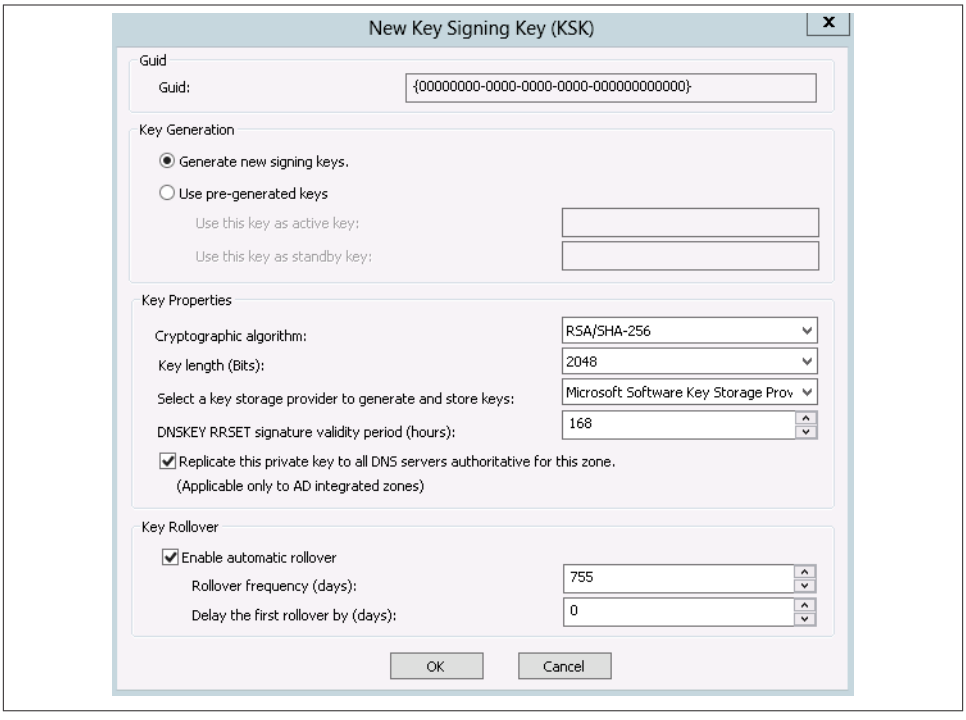


Figure 8-4. Generating a new key signing key

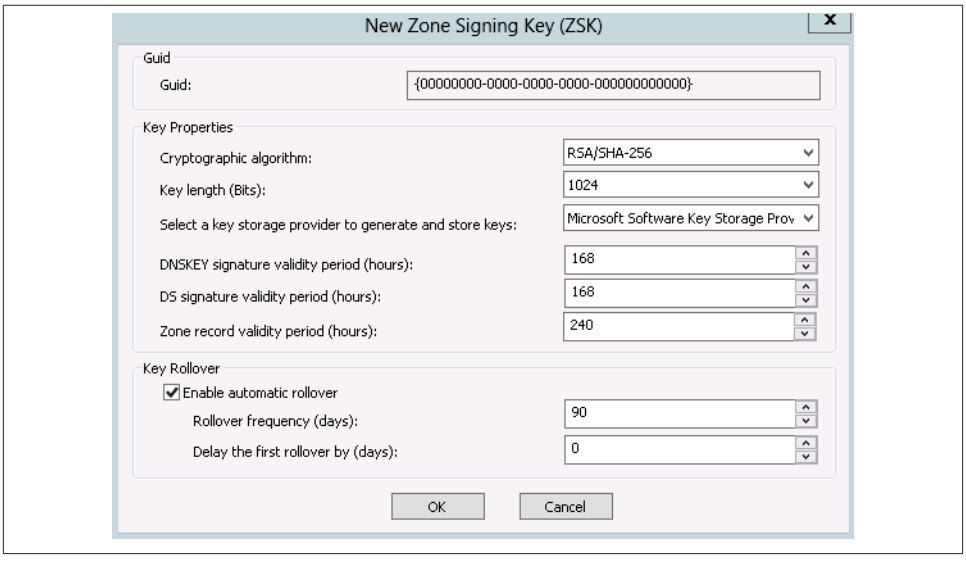


Figure 8-5. Creating a new zone signing key

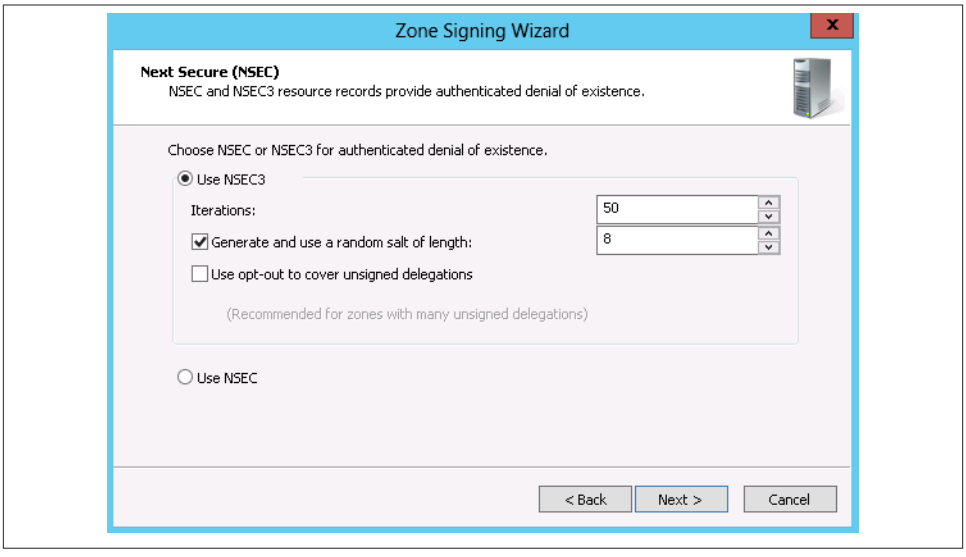
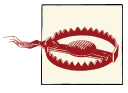


Figure 8-6. Next Secure record configuration

Finally, you will need to decide whether or not to publish the trust anchors for the zone to Active Directory, as shown in Figure 8-7. Once you publish the trust anchors, all DNS servers will begin validating responses for the zone. By default, trust anchor distribution is disabled.



Unless you are confident in the configuration you've completed and the configuration of the clients and other DNS servers in the environment, you will want to perform testing with a limited set of DNS servers and clients before deploying the trust anchors.

The wizard will provide you with a copy of the information that will need to be stored in the DS record in the parent zone on the subsequent screen. If you do not own the parent zone, and this was for a public DNS zone, you will need to provide the DS record data to the owner of the parent zone (often your domain registrar).

Finally, the wizard will provide a progress screen as the zone is signed for the first time. Once the wizard completes and you refresh the console, you'll see new records, similar to Figure 8-8. The DNS console also indicates that a zone is signed by overlaying a lock icon on the zone in the tree to the left.

If you want to change any of the properties of the DNSSEC configuration for the zone, you can access those SEC properties by right-clicking the zone and choosing

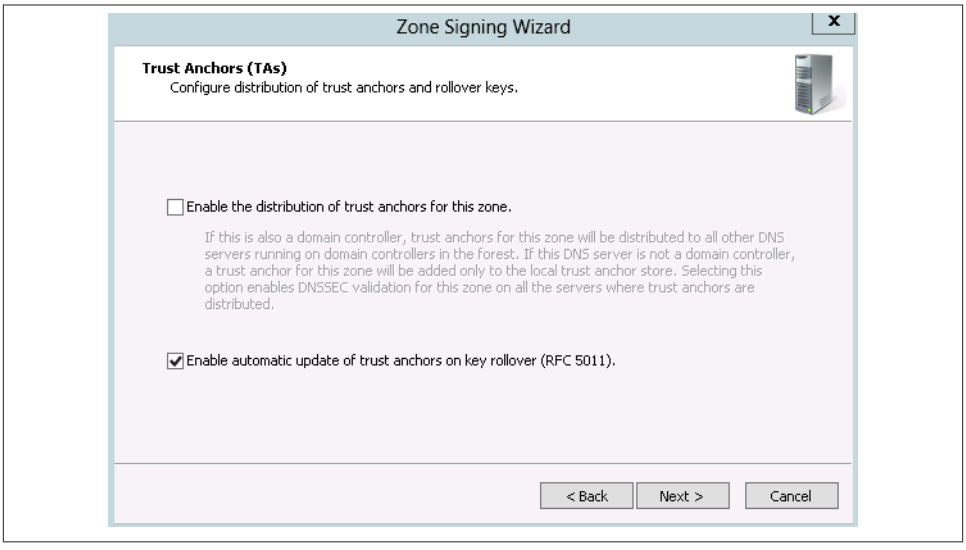


Figure 8-7. Configuring trust anchor distribution

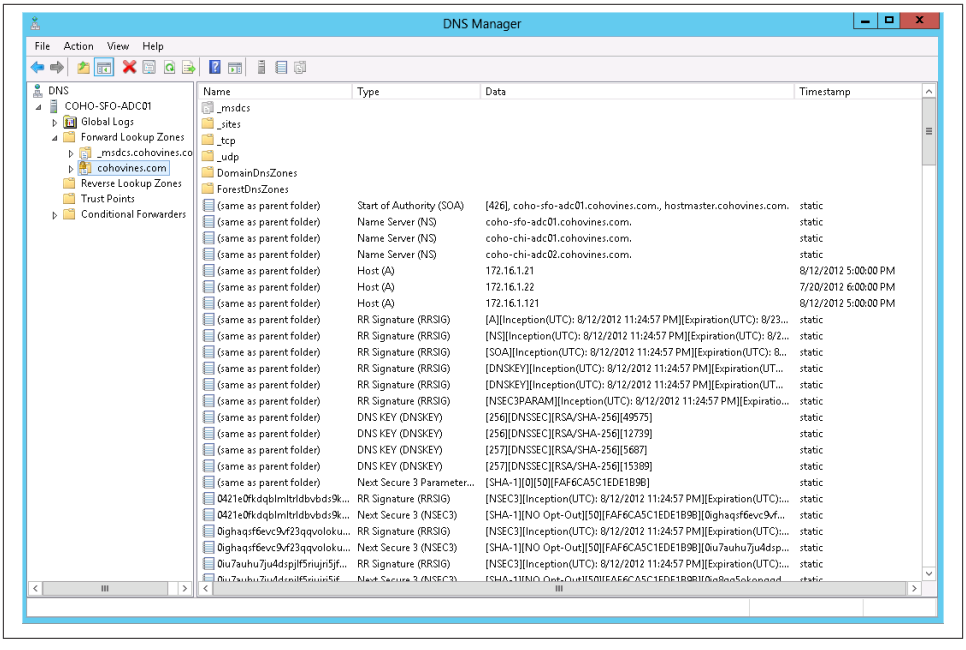


Figure 8-8. Signed DNS zone contents

DNSSEC→Properties. When you are ready to publish trust anchors, you can do so by checking "Enable the distribution of trust anchors for this zone," as shown in Figure 8-9.

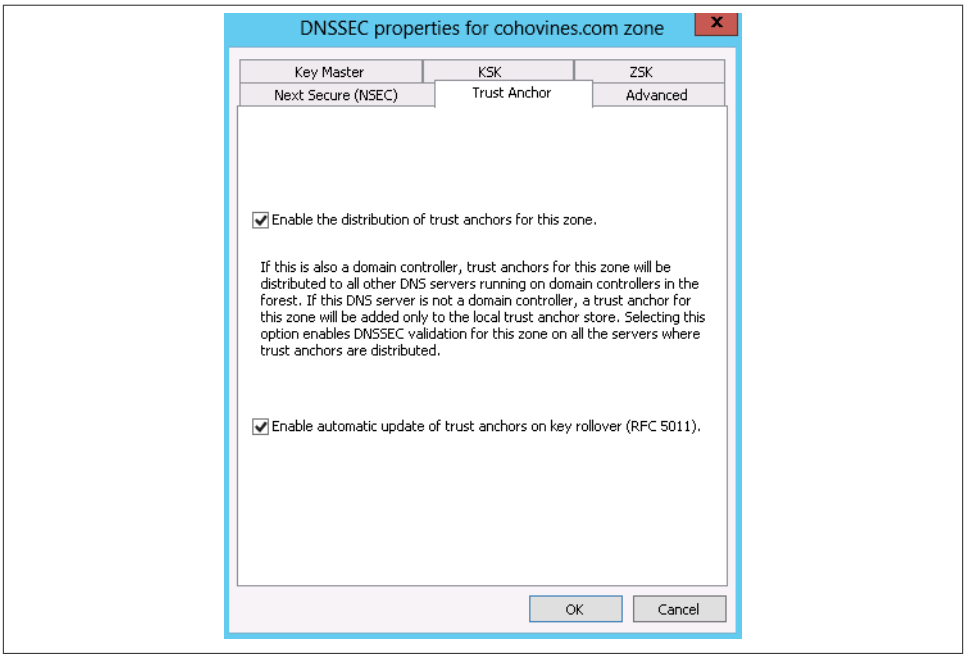


Figure 8-9. DNSSEC zone properties

DC Locator

One of the fundamental issues for clients in any 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 for the domain it is a member of:

1. A client previously located in Site A is moved to Site B.
2. When the client initially boots up, it thinks it is still in Site A, so it retrieves from the cache the last DC it used—most likely from Site A—and proceeds to contact it.
3. The DC in Site A receives the request, determines that the IP address is for Site B, and determines that the client should now be using a DC that services Site B. If the server does not service clients in Site B, it will return the client's new site in the reply and tell the client that it isn't the closest DC.

4. The client will then perform a DNS lookup to find all DCs that service Site B.
5. The client then contacts a DC servicing Site B. Three things can happen at this point:
   a. The DC servicing Site B can respond and authenticate the client.
   b. The DC might fail to respond (it could be down), at which point the client will attempt to use a different DC in Site B.
   c. All DCs servicing Site B might fail to respond, in which case the client will perform a DNS lookup to find all DCs that service the domain, regardless of what sites they cover, and use any one of them to authenticate.

The two main things that are needed to support an efficient DC locator process are proper definition of the site topology in Active Directory and the presence 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 Knowledge 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*.

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.
```

Although 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, similar to the preceding record.

Next, we have the following record:

```
ec4caf62-31b2-4773-bcce-7b1e31c04d25._msdcs.mycorp.com. 600 IN CNAME
 moose.mycorp.com.
```

This is an alias or canonical name (CNAME) record. It is contained under the *_msdcs* subdomain, which is used by domain controllers to intercommunicate. The record is comprised of the DSA GUID for the server, which is an alias for the server itself. DCs use this record if they know the DSA GUID of a server and want to determine its IP address.

CNAME records are aliases to other records. A CNAME can point to another A record, or even another CNAME. The benefit of using a CNAME is in the scenario where you need to point to one host with multiple names.

For example, if your web server is accessible at *www.mycorp.com* and you also provide FTP services from your web server, the IP addresses for the two services will always be the same. Instead of having two A records, *www* and *ftp*, you could have one A record, *www*, and a CNAME, *ftp*, that is aliased to *www*.

The advantage here is that if the IP address of your server changes, you will only have to update it in one place.

Next, we have this A record:

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

This is registered only if the domain controller is also a Global Catalog server. You can query *gc._msdcs.mycorp.com* with *nslookup* to obtain a list of all the Global Catalog servers in the forest.

The remaining resource records are of type SRV. The SRV record type was defined in RFC 2782, "A DNS RR for Specifying the Location of Services (DNS SRV)." The full text can be found at *http://tools.ietf.org/html/rfc2782*. Simply put, SRV records allow you to specify one or more servers on your network that should be used for specific protocols. These records also allow you to remap the port numbers for individual protocols or the priority of servers, determining the order in which certain servers are used.

Even if you change the port numbers for Active Directory-specific SRV records, Windows will not honor these new port numbers.

A few more GC-specific records are shown next:

```
_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.
```

One interesting thing to note about SRV records is the seventh field, which designates the port used to contact the service on that host. In all of these cases, 3268 is used, which corresponds to the Global Catalog port. You may have also noticed the entries that contain *Default-First-Site-Name*. Each Global Catalog server registers site-specific

records so clients can find the optimal Global Catalog based on their site membership. See the upcoming sidebar "Site Coverage" on page 190 for more information.

The next few SRV records are for Kerberos authentication (port 88) and the *kpasswd* process (port 464), which allows users to change passwords via Kerberos:

```
_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.
```

Site Coverage

You can create sites in the Active Directory site topology that do not have domain controllers located within them. The domain controllers that have the best connections as defined by the site links will "cover" for such a site. This functionality is known as *automatic site coverage*.

When a DC covers for a site, it will add site-specific SRV records that advertise it as a DC that can handle queries for clients in the site. To see a list of the sites that a particular DC is covering for, run the following *nltest* command, replacing *dc01* with the name of the DC for which you want to determine coverage:

```
c:\> nltest /dsgetsitecov /server:dc01
```

In some scenarios, you may find it necessary to disable the automatic site coverage behavior by toggling a registry value on your domain controllers. To disable automatic site coverage, create a registry value `AutoSiteCoverage` of type `REG_DWORD` under `HKLM\System\CurrentControlSet\Services\NetLogon\Parameters` and set it to **`0`**. Branch office deployments in particular are a common scenario for this. For more information on this configuration option, see this link.

Just as with the Global Catalog SRV records, there may be more site-specific Kerberos records for any additional sites the DC covers.

The rest of the SRV records are used to represent a domain controller for a particular domain and site:

```
_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.
```

One record to note is the *_ldap._tcp.pdc._msdcs.mycorp.com.* entry, which is registered by the DC that is acting as the PDC Emulator for the domain. No other FSMO roles are registered in DNS.

Based on all these records, you can obtain a lot of information about an Active Directory environment by doing simple DNS queries. Some of the information you can retrieve includes:

- All Global Catalog servers in a forest or a particular site
- All Kerberos servers in a domain or a particular site
- All domain controllers in a domain or a particular site
- The PDC emulator for a domain

Overriding SRV Record Registration

There may be times when you do not want domain controllers or global catalogs publishing some or all of their records outside of the site they are in, or maybe you do not want them publishing any records at all. For example, you may have domain controllers that should be dedicated to an application like Microsoft Exchange that runs in the site where the domain controllers are located, or domain controllers in branch offices, or you may have domain controllers that are dedicated to just replicating Active Directory for backups.

You have two options for configuring the SRV record registration behavior:

- Use the `DnsAvoidRegisterRecords` registry entry. For more details on configuring this option, see Microsoft KB article 306602.
- Use NetLogon system settings in the administrative templates of the group policy applied to domain controllers. For more details on configuring this option, see DNS Support for Active Directory Tools and Settings.

Regardless of which mechanism you choose, both give you the option of specifying which records should and shouldn't be registered. When determining which route to take, be sure to consider the manageability aspect in the event you need to change the settings later.

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 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.

Not Delegating the AD DNS Zones

The first impulse of any cost-conscious organization should be to determine whether their existing DNS servers can be authoritative for the AD zones. That could entail manually populating all the necessary resource records required by each DC if the current DNS implementation doesn't support dynamic updates. While this sounds fairly trivial at first glance, it becomes decidedly less trivial once you begin to explore the requirements.

Political factors

If the existing DNS servers are utilized for the AD DNS zones, the Active Directory administrators will likely not have the same level of control as they would if the zones were delegated and managed by them. This separation of duties—and lack of control over a critical dependency like DNS—may make it difficult for AD administrators to guarantee a certain level of service for AD.

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. For example, domain controllers try to register their resource records via DDNS on a periodic basis. Many 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.

Non-Windows DDNS implementations often restrict which clients can perform DDNS registrations and updates based on IP address. With this in mind, DNS administrators will need to configure DDNS to only allow domain controllers to update certain zones in order to mitigate the security risks of allowing too many clients the ability to update any DNS record in the server. This should not typically be a problem, at least at a technical level, but you will need to ensure that you communicate changes and plan for turnaround time when adding new domain controllers or changing the IP address of a DC.

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 domain controllers 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). In addition, promoting one DC could cause other DCs to update the records they need registered in DNS as well.

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.

Integration issues

When Windows 2000 Active Directory was first released in 1999, this was more of a problem than it is today, but older versions of DNS server or administration software may not support SRV records or underscores in zone or record names (e.g., *_msdcs.mycorp.com*). Upgrading to the latest version of your DNS server platform should be a priority in this case.

Figure 8-10 shows how the client request process is straightforward when the AD DNS zones are not delegated. Clients point at the same DNS servers they always have.

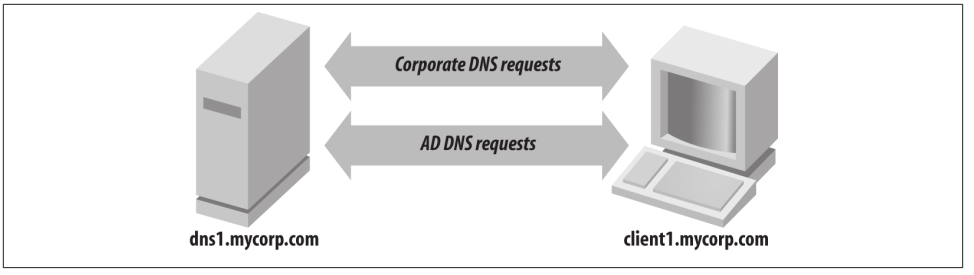


Figure 8-10. Active Directory and DNS

Delegating the AD DNS Zones

While at first glance it may seem pretty straightforward to support AD DNS zones in your existing DNS infrastructure, it can cause difficulties depending on your environment. Perhaps the most straightforward option is simply to delegate the DNS namespace(s) Active Directory will use to your domain controllers and allow them to host the DNS zones. If you use AD-integrated DNS zones, the maintenance becomes even easier. After you've done the initial creation of the zones by promoting a domain controller, the records are stored in Active Directory and distributed to the other domain controllers via replication.

Political factors

Frequently, organizations will have a central DNS team that manages and supports name resolution. If you make the decision to delegate the AD DNS zones to domain controllers, for example, a significant part of name resolution for your clients will not be done on the existing corporate name servers anymore. This can make the DNS administrators uncomfortable, and rightly so. Note that Active Directory does allow you to delegate the management of records in an Active Directory-integrated DNS zone to other administrators very easily.

Initial setup and configuration

The initial setup to delegate the AD DNS zones is straightforward. An NS record and any necessary glue records—for example, an A record for the server to which you're delegating—need to exist on the parent zone pointing to the servers that will be authoritative for the zones. The rest of the configuration must be done on the servers that are going to support the AD DNS zones. If those servers are one or more domain controllers running Active Directory-integrated DNS, you will only need to add the DNS service (newer versions of Windows will allow you to perform this step as part of the DC promotion process).

While it is entirely possible to run the DNS zones supporting Active Directory on Windows-based DNS servers other than the domain controllers, it is unlikely that this will make much sense considering the enhanced functionality that is only available when running Windows DNS on a domain controller.

Support and maintenance

Ongoing support and maintenance of the AD DNS zones is very minimal, especially if you are using AD-integrated zones. In fact, since the domain controllers can use DDNS to update DNS, this is one of the primary benefits of using this method.

Integration issues

Unless you already run Windows DNS server, it is unlikely you'll be able to manage the AD DNS zones in the same manner as your existing DNS infrastructure. Figure 8-11 illustrates how, by delegating the AD DNS zones, you can still have clients point to the same DNS servers they do today. A variation of this approach would be to point the clients at the AD DNS servers and configure forwarding, as described in the next section.

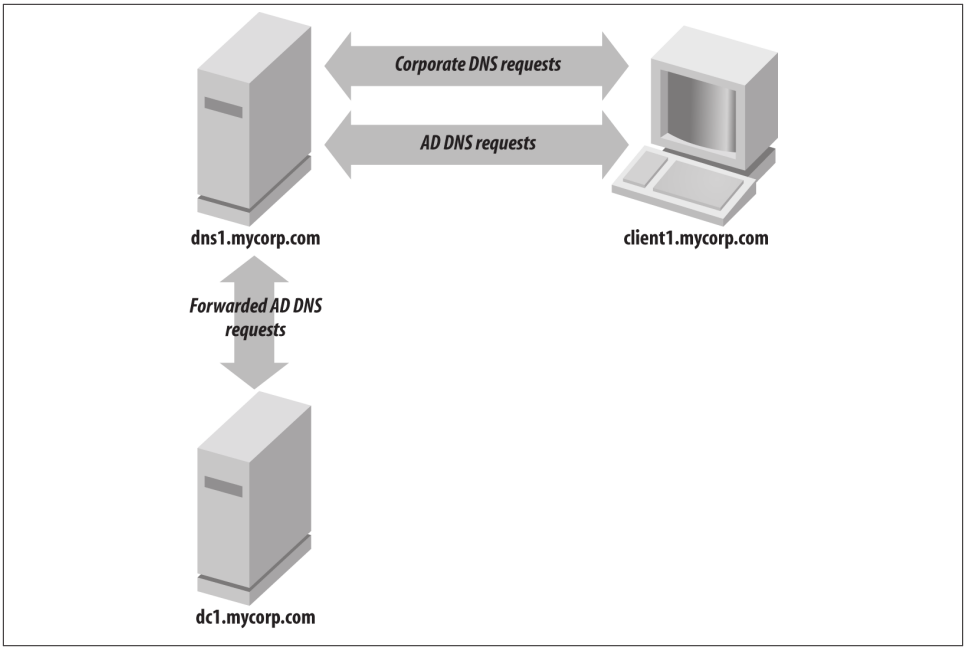


Figure 8-11. Client request flow when delegating the AD DNS zones

Active Directory-Integrated DNS

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

Conditional Forwarding

Conditional forwarding is a feature available in Windows Server 2003 and newer that gives administrators much more flexibility in how forwarding is handled than was available under Windows 2000. Figure 8-12 shows the forwarders configuration screen in the Windows Server 2003 DNS MMC snap-in. It allows you to set up one or more IP addresses to forward all requests that cannot be handled by the local DNS server. As you can see, we configure forwarding based on the domain name being queried:

- If the query is for *foobar.com*, forward to 10.1.1.1.
- If the query is for *example.com*, forward to 10.1.2.1.
- If the query is for any other zone, forward to 10.1.3.1.

Conditional forwarding allows you to create a more efficient resolution topology, particularly for corporate networks using disjointed namespaces, by sending queries directly to servers responsible for the individual zones instead of using only recursive queries to the Internet.

Windows Server 2008 greatly improved upon the conditional forwarders functionality by supporting the replication of conditional forwarder definitions via Active Directory. With Windows Server 2003, you need to configure conditional forwarders on every DNS server individually, which, in a large environment, is a tedious task that you will likely need to automate with a script.

The conditional forwarders configuration is now prominently available in the DNS MMC console at the same level as the folders for the forward and reverse lookup zones. All of the conditional forwarders defined on the DNS server are shown here, and you can also create new conditional forwarders via the right-click context menu. Figure 8-13 shows the new Edit Conditional Forwarder dialog. Notice the Active Directory replication scope option at the bottom of the screen.

In the normal world of DNS, you have two types of name servers: *primary* (sometimes called a master) 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 the primary for each

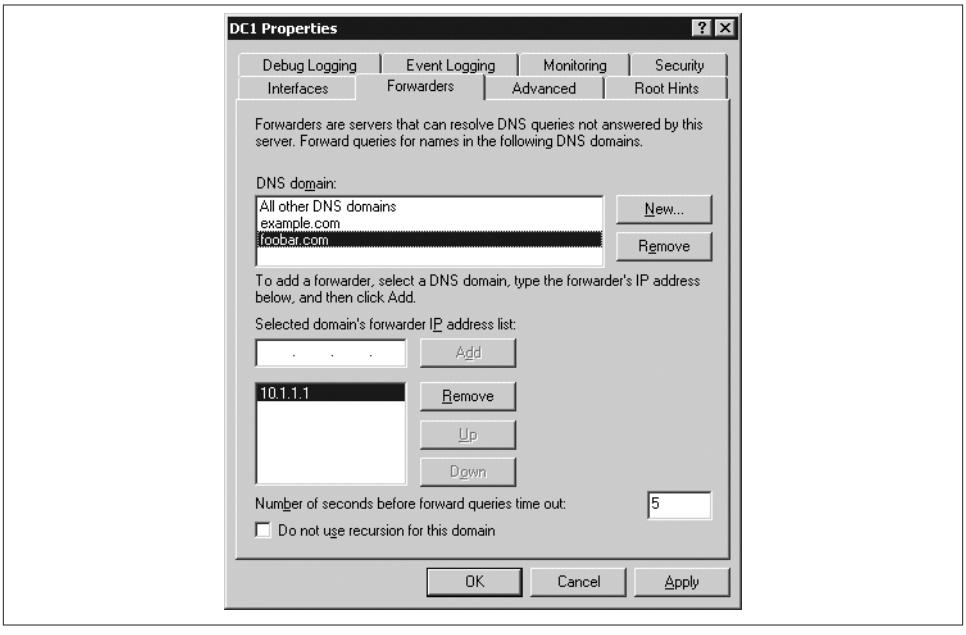


Figure 8-12. Forwarders configuration screen in the Windows Server 2003 DNS MMC snap-in

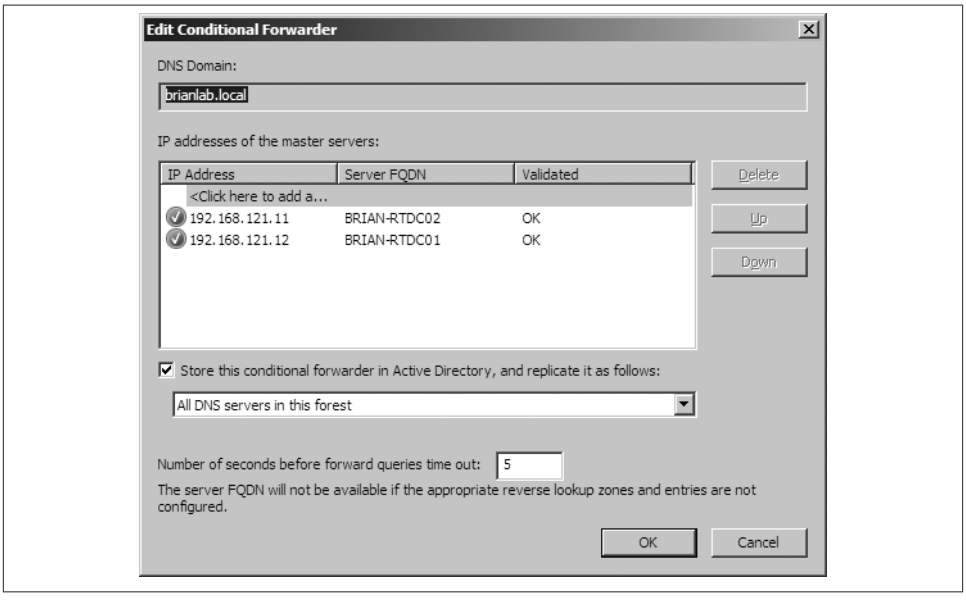


Figure 8-13. Managing a conditional forwarder in Windows Server 2008

zone and requests a copy of the relevant zone file 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 for the zones that it hosts. 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.

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 writable 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.

There is a possible issue when using integrated DNS with Windows 2000 domain controllers, called the "DNS island" issue. Active Directory requires proper DNS resolution to replicate changes, and when using integrated DNS, the domain controllers replicate DNS changes through Active Directory replication. This is the classic chicken-and-egg problem; it can be avoided with proper configuration. The issue occurs when a forest root domain controller configured as a name server points at itself for DNS resolution. If that DC changes its IP address, the DNS records are successfully updated locally. However, unless the other DCs point at that same DC for their DNS resolution, they cannot resolve the DC's IP address, so replication fails and no other DC gets the IP address change. The forest root domain controller has effectively been segregated from the rest of the forest and becomes its own replication island. To avoid this issue, the forest root domain controllers that are name servers should be configured to point at root servers other than themselves.

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 were stored in the System container for a domain. That meant that every domain controller in that domain would replicate that zone data regardless of whether the

domain controller was a DNS server or not. For domain controllers that are not DNS servers, there is no benefit to replicating the data. Fortunately, there is a better alternative in newer versions of Windows: using application partitions, as described in the next section.

If you initially created your AD-integrated DNS zones under Windows 2000, they may still be replicating via the System container in the domain.

Background Zone Loading

A new feature added in Windows Server 2008 is background loading of DNS zones. Prior to Windows Server 2008, the DNS server service would not become available until it completed loading all of the zones it hosted from Active Directory. On a DNS server with large zones, loading all of the zones can take quite some time (hours, in some cases). With background loading, the DNS server service no longer waits until every zone is loaded but instead loads them in the background and makes the zones available for query/update as they complete the loading process.

This background loading behavior is only applicable to zones stored in Active Directory. Zones stored on the filesystem are still loaded serially prior to the service being available to service requests.

Using Application Partitions for DNS

Application partitions, as described in Chapter 4, are user-defined partitions that have a customized replication scope. Domain controllers that are configured to host 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, so long as they are in the same forest. 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 it 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.

There is one default application partition for each domain and forest. When configuring an AD-integrated zone, you have several options for storing the DNS data. These options are listed in Table 8-3.

Table 8-3. 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 was 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=CampusDnsZones,dc=mycorp,dc=com`	To domain controllers that have been enlisted to replicate the application partition.

The final option in Table 8-3 highlights that you can define custom application partitions outside of the default *DomainDnsZones* and *ForestDnsZones* partitions. Some organizations elect to define additional application partitions to control the replication scope of other DNS zones. For example, your domain controllers might be hosting DNS zones specific to a certain building or campus, so you might define an application partition called *CampusDnsZones* just to host these zones. You would, in turn, configure only the domain controllers at that campus to host this application partition, thereby limiting the replication scope of those zones.

When installing a new Windows Server 2003 Active Directory forest, the default DNS application partitions are created automatically. If you've upgraded from Windows 2000, you can manually create the DNS application partitions by using the DNS MMC snap-in or the *dnscmd.exe* utility. To manually create the DNS application partitions with the DNS MMC snap-in, right-click the server on the lefthand side and click "Create Default Application Directory Partitions." Once you have created the application partitions, you must modify the replication scope of each zone individually to take advantage of the application partition replication, using these steps:

1. Right-click the zone you want to update and click Properties.
2. Click the "Change" button next to the Replication section of the General tab.
3. Select "To all DNS servers running on domain controllers in this domain" (*DomainDnsZones*) or "To all DNS servers running on domain controllers in this forest" (*ForestDnsZones*), as shown in Figure 8-14.

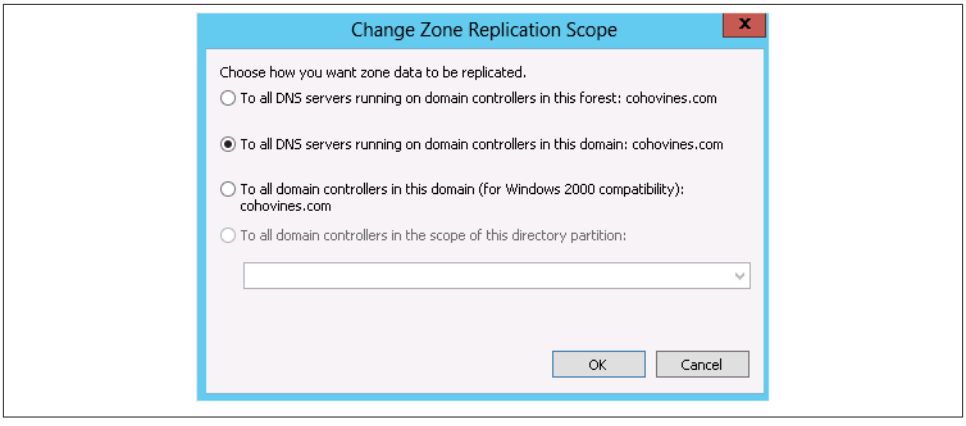


Figure 8-14. Modifying a DNS zone's replication scope

Aging and Scavenging

One of the complexities of running a DNS infrastructure where clients can register their own records is that over time, records will build up in your zones for clients that no longer exist, or perhaps have a new name. If you only have a small number of machines on your network, you could likely manage these records yourself through the MMC. On the other hand, if you have hundreds or even hundreds of thousands of machines participating in a single DNS zone, chances are you won't be able to manage the lifecycle of these records on your own.

Fortunately, the Windows DNS implementation includes a feature called *scavenging* that can help. Scavenging is a background process that you configure on a per-DNS-server basis to scan all of the records in a zone and remove the records that have not been refreshed in a certain time period. Clients that register themselves with dynamic DNS will automatically refresh their DNS registrations periodically. Windows DNS will store this timestamp as an attribute of the DNS record. By default, Microsoft DNS clients update and refresh their DNS registrations once every 24 hours.

One of the scavenging configuration options is the "no-refresh interval," which defines how often the DNS server will accept the DNS registration refresh and update the DNS record. This functionality limits the amount of unnecessary replication for DNS record timestamp updates. For records that are manually created, the refresh timestamp is set to zero, which excludes the record from scavenging. You can also edit dynamically registered records to exclude them from scavenging.

Configuring Scavenging

Configuration of scavenging is a two-step process:

1. Enable scavenging for a specific DNS zone and define the refresh intervals for that zone.

If you right-click on the DNS server node in the DNS MMC, there is an option called "Set Aging/Scavenging for All Zones" that will allow you to configure these two values once for all of the DNS zones hosted on that particular DNS server.

2. Enable scavenging on at least one DNS server hosting the zone and define how often the scavenging process runs.

Be sure that you complete both steps. Many administrators forget one of these steps and discover that scavenging is not working as expected. You only need to complete step 2 on at least one DNS server hosting each zone enabled for scavenging. Some administrators opt to configure every DNS server for scavenging (step 2), while others opt to only enable the scavenging process on certain domain controllers. Generally it is easier to simply enable the process on each DNS server.

Setting zone-specific options

The options to enable scavenging for a particular zone are accessible via the Aging button on the properties of the zone (see Figure 8-15). As shown in the next section, "Enabling scavenging on the DNS server" on page 202, there are two configuration items of interest:

No-refresh interval
: This is how often the DNS server will propagate a timestamp refresh from the client to the directory or filesystem. The default value is seven days and is generally acceptable. The purpose of this setting is to reduce the amount of unnecessary replication or zone transfer activity that would be solely necessary for a timestamp update.

Refresh interval
: This is how long the DNS server must wait following a refresh for a record to be eligible for scavenging. The default value here is again seven days. In practical terms, if you use the default values here, and a client refreshes its record on the seventh day of the month and then goes offline, the DNS server will scavenge the record no earlier than the fourteenth day of the month.

Enabling scavenging on the DNS server

The actual scavenging process must be enabled and configured on one or more DNS servers for scavenging to occur. It is a common misconception that the only scavenging configuration necessary is on the DNS zone itself.

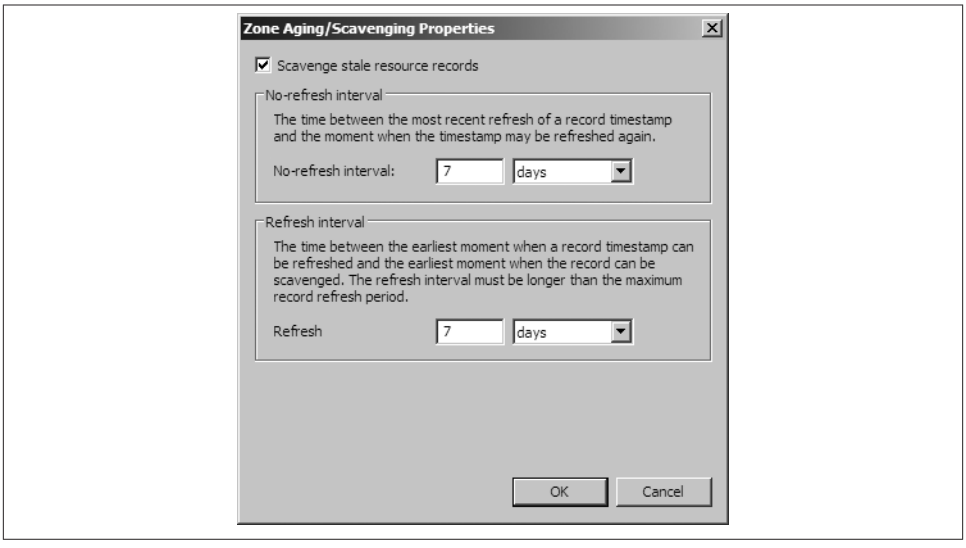


Figure 8-15. DNS zone scavenging settings

To change the DNS server scavenging settings, access the properties of the DNS server in the DNS MMC and choose the Advanced tab, as shown in Figure 8-16. You will need to enable the scavenging process and define an interval for how often it runs. The default of seven days is often acceptable; however, you can adjust it to run as frequently as every hour. Be careful that you don't set it to run so often that the DNS server is constantly running the scavenging process, though, as this would be counterproductive.

Managing DNS with Windows PowerShell

Traditionally, scripting common tasks involving Windows DNS servers has been difficult to do without a mix of complex tools. The DNS service has exposed various APIs via Windows Management Instrumentation (WMI) that you can access from VBScript and other scripting languages. Some tasks can only be automated with the *dnscmd* command-line tool, which requires complex text scraping and processing.

Windows Server 2012 introduces a wide range of Windows PowerShell cmdlets for managing DNS on Windows Server 2008 and newer DNS servers. In fact, with Windows Server 2012, you should be able to complete nearly any task *dnscmd* supports with Windows PowerShell. To make the DNS server cmdlets available and get a complete list of the cmdlets available for managing DNS, you can use these commands:

```
Import-Module DnsServer
Get-Command -Module DnsServer
```

Figure 8-16. DNS server advanced properties

In addition, the *DnsClient* and *DnsConfig* PowerShell modules provide the cmdlets necessary to manage the DNS configuration on a machine and resolve names. The *Resolve-DnsName* cmdlet provides the functionality that traditionally required you to use the *nslookup* command-line tool.

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 difficult, with each option having its own advantages and disadvantages. If you delegate the zones to domain controllers, AD-integrated zones can save you a lot of time in maintenance and upkeep and optionally enable you to divide the responsibility for DNS between traditional DNS administrators and the AD administrator.

CHAPTER 9

Domain Controllers

This chapter focuses on the infrastructure cornerstone of Active Directory—the domain controller. Domain controllers are the component that hosts all of the Active Directory functionality and protocols. In this chapter, we'll dig into the steps necessary to deploy (promote) new domain controllers as well as operational concerns around physical security and hardware virtualization.

Ensuring the physical security of Active Directory domain controllers is an incredibly important part of a successful, secure deployment of AD. As AD administrators are well aware, in many organizations—especially large global enterprises—the locations that host servers in branch offices are often far from secure. The *read-only domain controller* (RODC) is the Active Directory solution to this problem.

Hardware virtualization is nothing new, and in Windows Server 2012, Microsoft addressed many of the technical challenges and also simultaneously introduced the ability to rapidly clone domain controllers to scale out or meet disaster recovery requirements.

Building Domain Controllers

One of the first steps you're going to take when you deploy Active Directory is building domain controllers. Depending on the type of domain controller you're deploying and how you go about it, the process may vary quite a bit. You might want to deploy an RODC, or perhaps you want to use Install from Media (IFM) to lessen the replication load on the network and rapidly promote domain controllers. The process of converting a member server to a domain controller is known as *promotion*.

Deploying with Server Manager

The first step is to install the Active Directory binaries on the server being promoted to a domain controller. You can do this in Server Manager by clicking "Add roles and features" on the Dashboard, as shown in Figure 9-1. Continue through the wizard and select the server on which you want to install the Active Directory binaries, as shown in Figure 9-2.

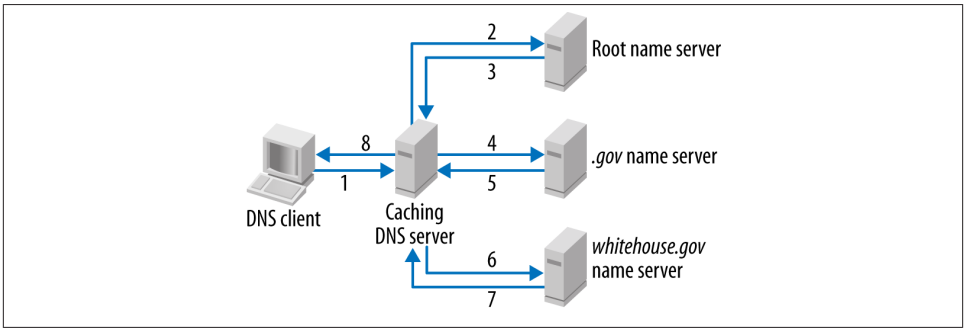


Figure 9-1. Server Manager Dashboard

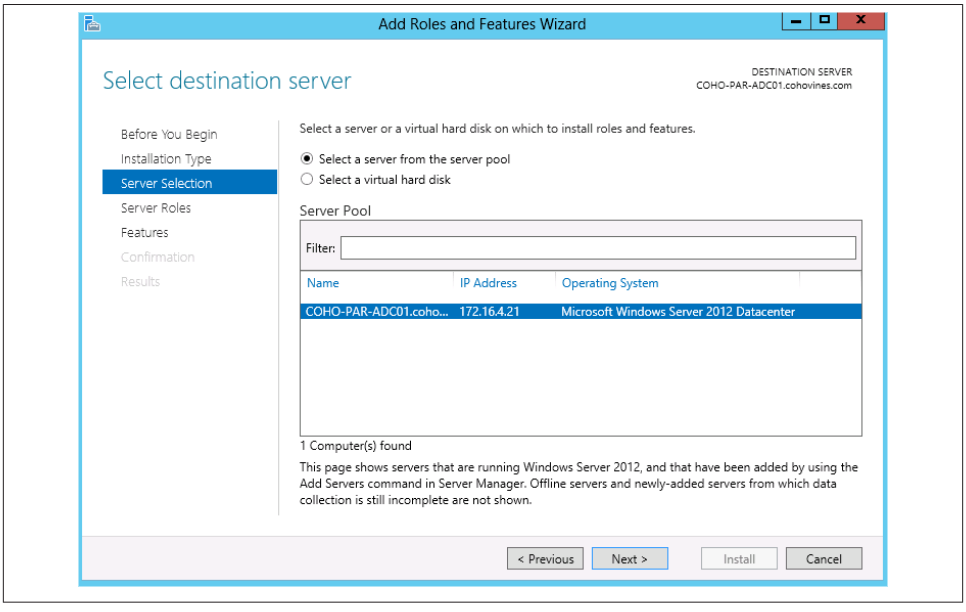


Figure 9-2. Select the destination server

On the "Select server roles" screen, select Active Directory Domain Services, as shown in Figure 9-3. Server Manager automatically locates the dependencies of the selected role and prompts you to confirm that you want to install the dependencies, as shown in Figure 9-4. Proceed through the rest of the wizard and click Install to complete the process.

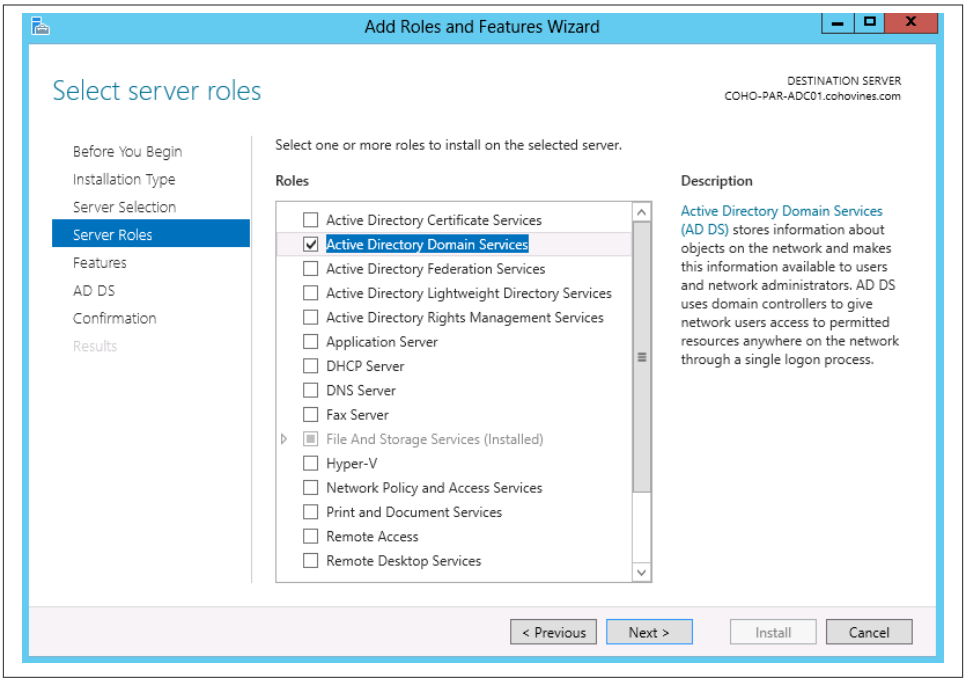


Figure 9-3. Select server roles to install

Once the installation completes, click the "Promote this server to a domain controller" link in the status box of the Results window. This will launch the Active Directory Domain Services Configuration Wizard. If you're familiar with promoting domain controllers in versions of Windows prior to Windows Server 2012, this wizard replaces *dcpromo*. If you launch *dcpromo* on Windows Server 2012, you'll get the error shown in Figure 9-5.

In this example, we'll add an additional domain controller to an existing domain, as shown in Figure 9-6. If you were creating a new forest (or domain) from scratch, you'd select the relevant option in Figure 9-6 in lieu of the default "Add a domain controller to an existing domain." All of the steps are the same, apart from a couple of additional questions that the wizard will ask.

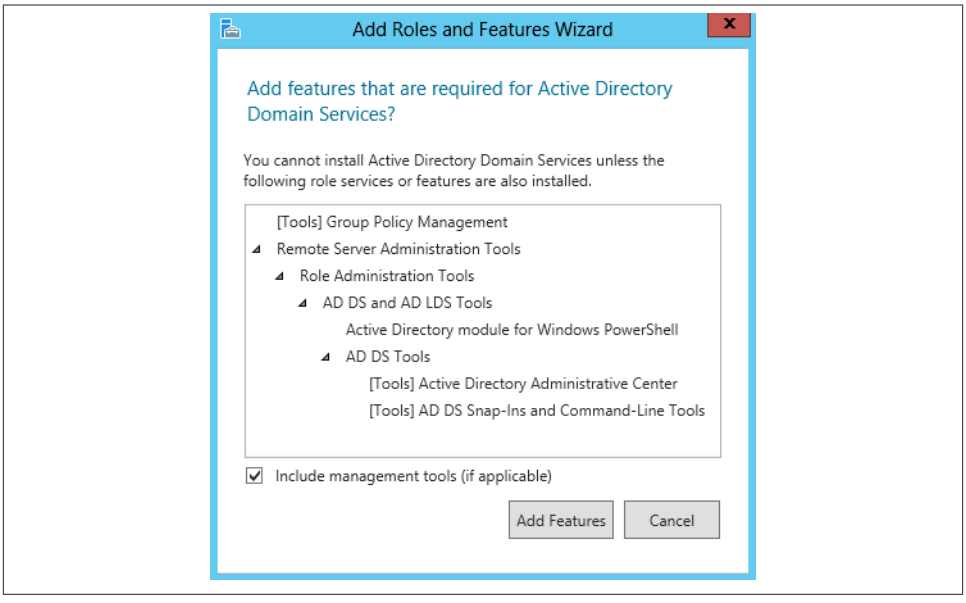


Figure 9-4. Selected role dependencies

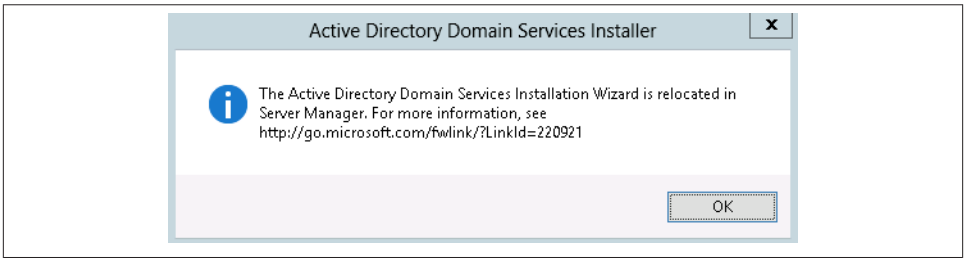


Figure 9-5. DCPromo error in Windows Server 2012

At the screen shown in Figure 9-6, you'll need to provide credentials with enough access to promote the new domain controller. Generally speaking, this is a domain admin account in the target domain. If you're going to deploy an RODC, though, you can use the credentials that were used to prestage the RODC's computer account in Active Directory.

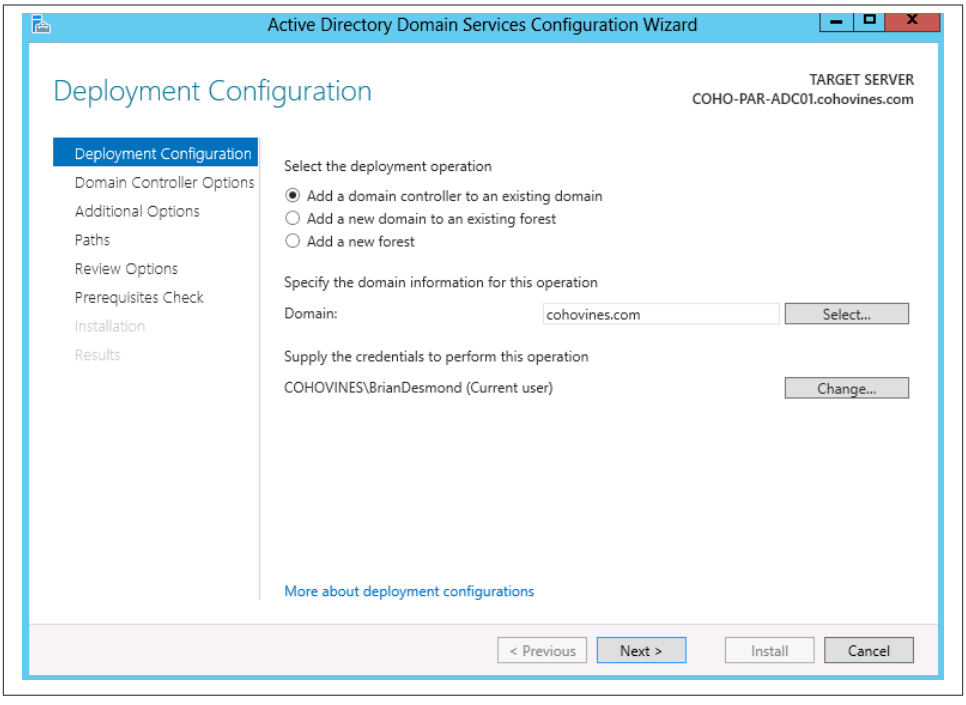


Figure 9-6. Beginning the AD DS Configuration Wizard

Next, the wizard will allow you to define key characteristics of the domain controller. These include the Active Directory site the DC will service and the Directory Services Restore Mode password (discussed in Chapter 18). Of particular interest (as shown in Figure 9-7) are whether or not this DC will host DNS, if it will be a Global Catalog, and whether or not it will be an RODC. In this example, we'll promote a Global Catalog into the Paris site.

In the DNS Options step, you may be warned that a delegation for the DNS zone cannot be created, as shown in Figure 9-8. In general, this is safe to ignore. If you are creating a new domain or forest, this step can help you set up DNS by creating a delegation in the parent DNS zone for you. We discussed DNS delegations in Chapter 8. If you skip this step for a new domain or forest, you'll simply need to create a new delegation if it becomes necessary.

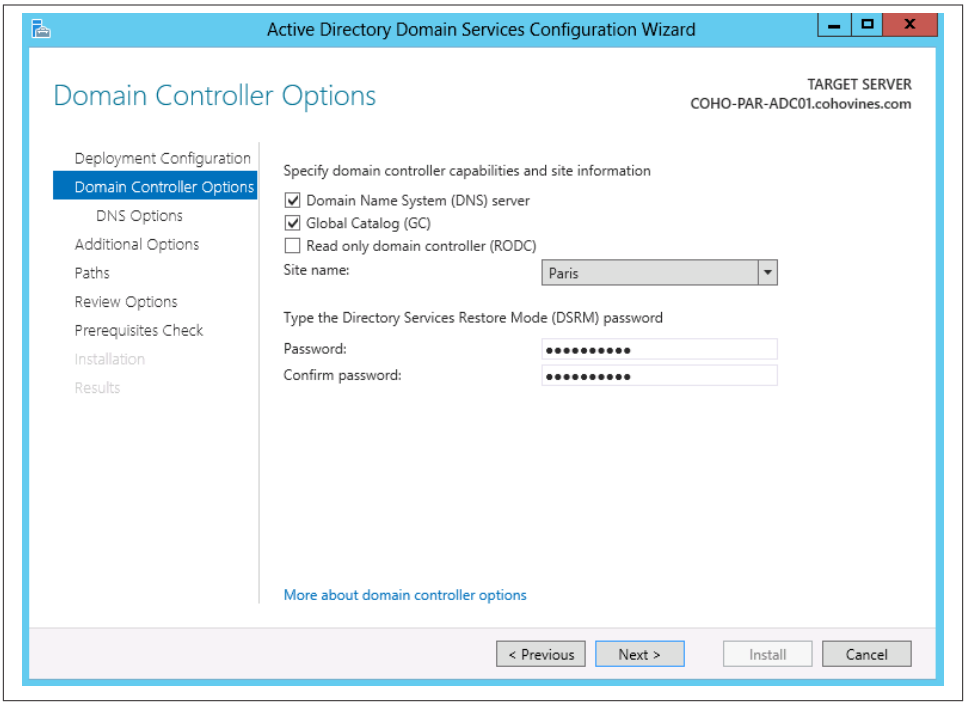


Figure 9-7. The Domain Controller Options screen

Next, the Additional Options screen shown in Figure 9-9 lets you choose the source domain controller for initial replication. This can be useful if you need to control network impact or load on other domain controllers as part of the initial replication process. You can also opt to use Install from Media to speed up promotion. We discuss IFM in depth in Chapter 18.

On the next screen (Figure 9-10), you will be able to specify the paths to the Active Directory database, transaction logs, and Sysvol share. In this example, we'll keep the defaults, but this is a topic of much debate. In general, unless you are running a large, complex Active Directory environment, you'll probably be fine sticking with the defaults or simply placing all of the paths on an alternate drive if your policy is not to use the system drive for data.

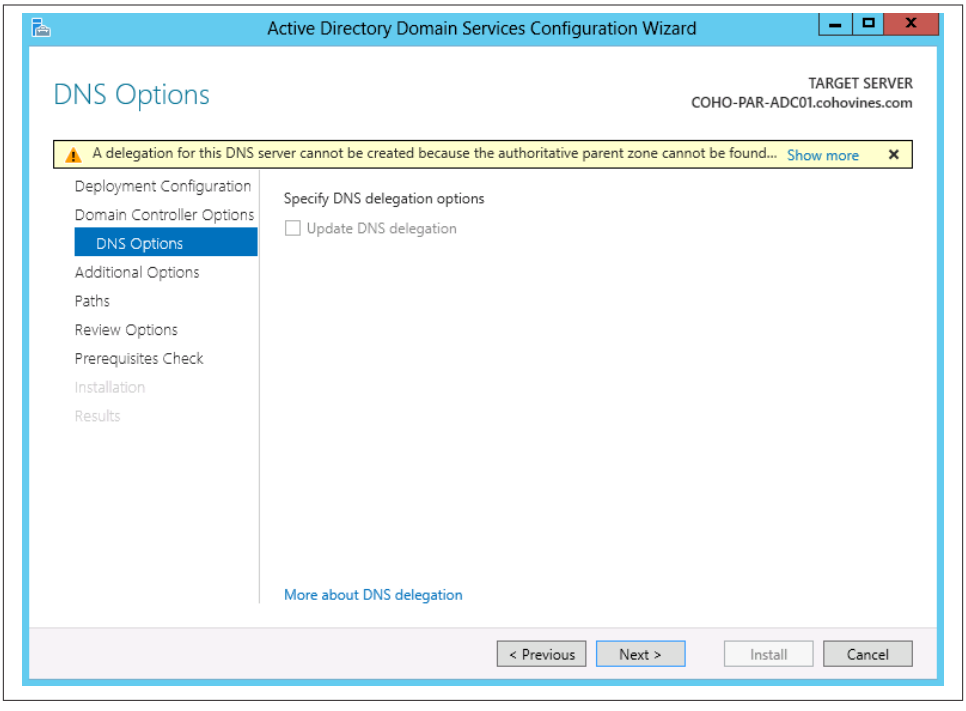


Figure 9-8. DNS delegation warning

If you're running a larger environment, you may want to separate these three items. The Active Directory database is primarily a read-intensive workload. In general, with 64-bit domain controllers, if you can cache the entire database file in RAM you will really only incur a large read impact on your storage when the domain controller starts up and hasn't cached any data. If you are planning storage for the database, make sure to account for both the size of the database and whether or not the Redundant Array of Inexpensive Disks (RAID) array can accommodate the read workload. RAID types such as RAID-5 and RAID-10 are usually well suited for this type of workload.

Most Active Directory environments don't incur a proportionally high degree of changes to the directory. When changes are made, however, they are first written to the transaction logs. The transaction logs will never take much space, but in a very busy environment they will demand storage that can keep pace with write requests. RAID types such as RAID-1 and RAID-10 are generally well suited for the transaction logs.

Finally, the Sysvol share is primarily a read-intensive volume. The Sysvol share can take quite a bit of space in environments that use Group Policy heavily. The advent of the central store feature for ADMX files has helped mitigate this quite a bit, but you should still plan storage accordingly. The larger question to consider is what the impact of the

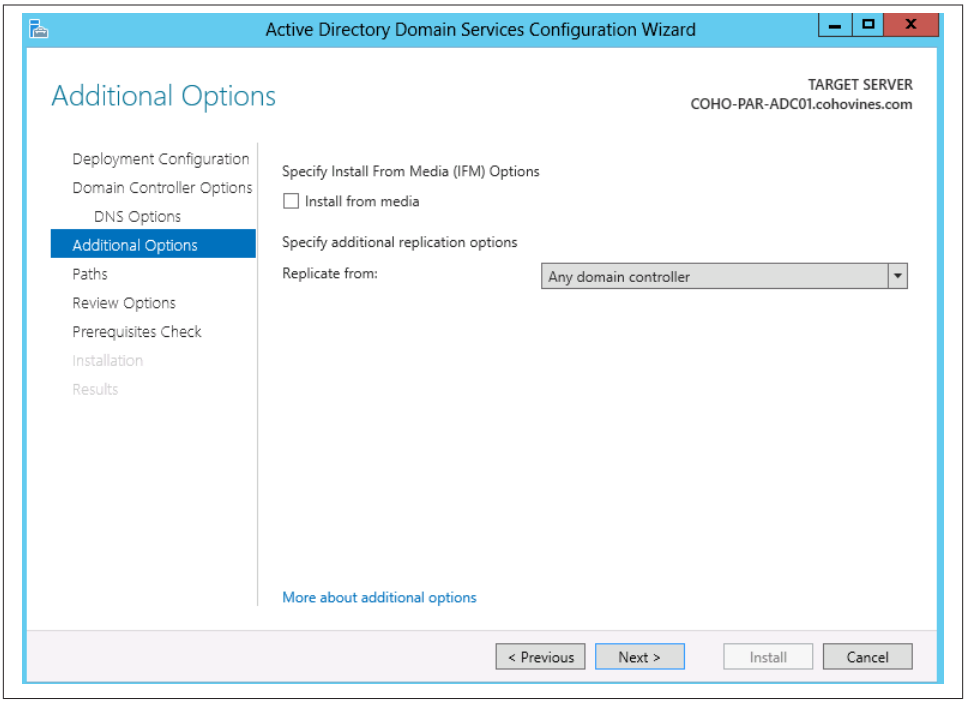


Figure 9-9. Replication and IFM settings

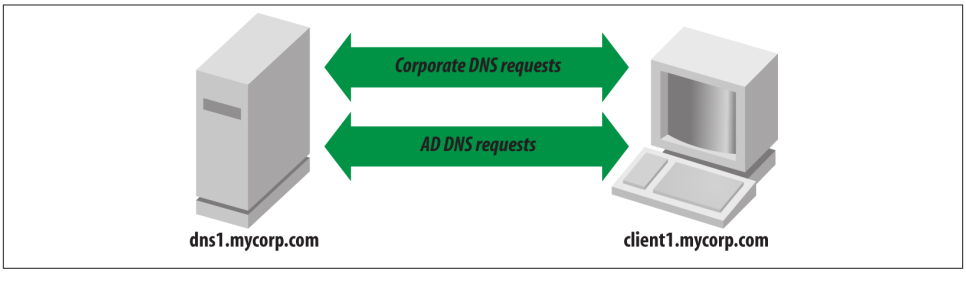


Figure 9-10. Data storage paths

Sysvol share could be on the overall availability of your Active Directory domain. When you delegate control of a GPO, you are also giving that administrator rights to store data in Sysvol in the GPO's share. If the delegated administrator were to place so much data in Sysvol that it filled up the drive hosting Sysvol, you could be in for trouble. The scenario where you might have an issue is if Sysvol is on the same drive as your OS installation, or the Active Directory database or transaction logs. If you're going to

delegate access to edit group policies, consider putting Sysvol onto a separate partition or volume.

On the Review Options screen shown in Figure 9-11, the wizard summarizes your choices in an easy-to-read format. Perhaps substantially more interesting, though, is the "View script" button at the lower-right corner of the screen. This button provides you with the PowerShell command necessary to promote the domain controller using the options you selected in the wizard. Instead of trying to figure out all of the options to promote DCs with PowerShell, you can run through the wizard once, gather the script, and then use it to build multiple DCs quickly and easily.

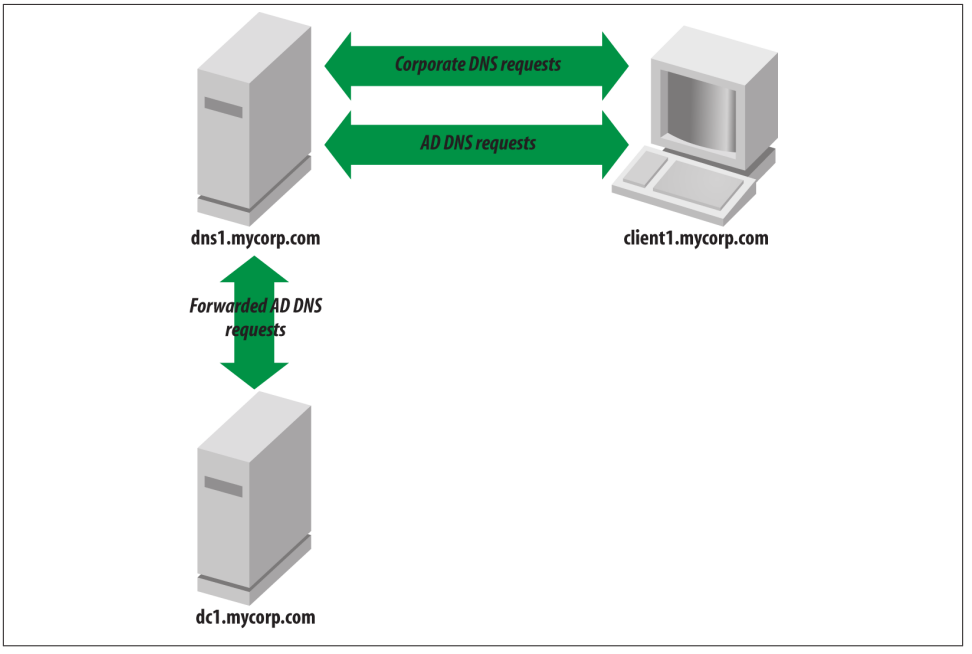


Figure 9-11. Confirmation and PowerShell script export

The wizard will check prerequisites, and if your server passes (as shown in Figure 9-12), you can click Install. Once you click Install, Active Directory will configure the server to be a domain controller, perform initial replication, and then reboot automatically. After the server reboots, you will have a fully functional domain controller.

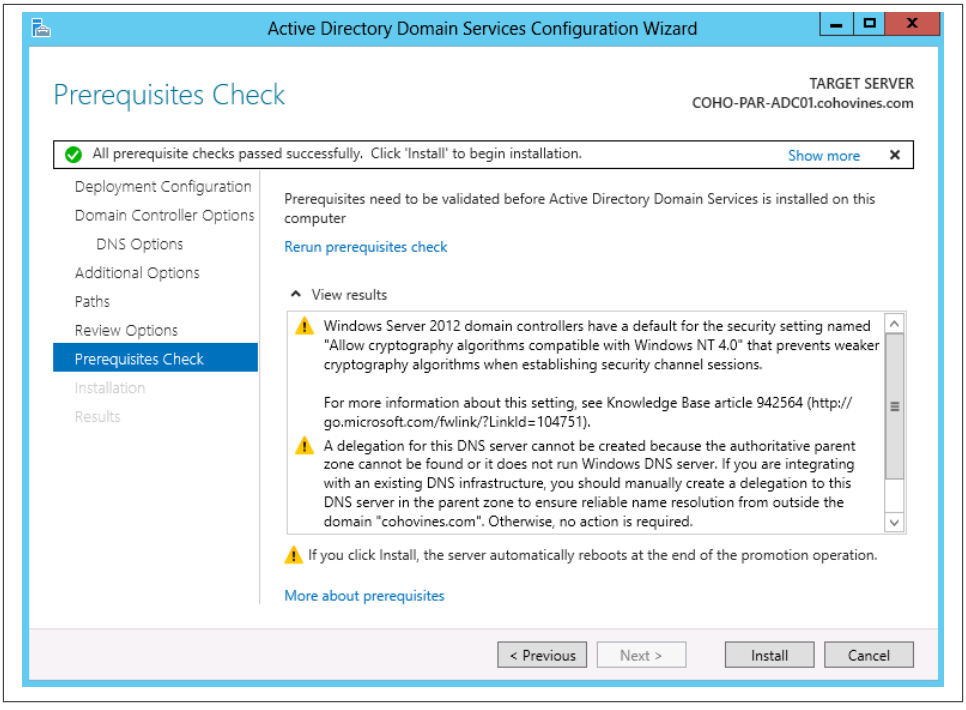


Figure 9-12. Final confirmation

Using DCPromo on Earlier Versions of Windows

If you're promoting a domain controller that's running a version of Windows prior to Windows Server 2012, the Server Manager and PowerShell approaches we've discussed so far won't apply. Prior versions of Windows (dating back to Windows 2000) use a tool called *dcpromo* to install Active Directory. You can launch *dcpromo* by going to Start→Run and typing in **`dcpromo`**. If you review the screenshots and information in the previous section, "Deploying with Server Manager" on page 206, you'll find that the questions you get asked by *dcpromo* are nearly identical to the questions Server Manager asks.

One tip we can offer for *dcpromo* is that in some versions of Windows, options for things like Install from Media are hidden unless you run *dcpromo* in advanced mode. To do that, run *dcpromo /adv*.

Automating the DC Build Process

If you're installing lots of domain controllers or you just want to be consistent, you can automate the build process using Windows PowerShell starting with Windows Server

2012. Both the installation of the Active Directory server role as well and the promotion process can be easily automated.

The first thing you'll want to do is install the AD Domain Services server role if it's not already installed. You can do that with this PowerShell command:

```
Add-WindowsFeature -Name AD-Domain-Services -IncludeManagementTools
```

Next, you can promote the domain controller. In the following PowerShell command we promote the domain controller according to the same parameters chosen in the earlier section "Deploying with Server Manager" on page 206:

```
Install-ADDSDomainController `
-NoGlobalCatalog:$false `
-CreateDnsDelegation:$false `
-CriticalReplicationOnly:$false `
-DatabasePath "C:\Windows\NTDS" `
-DomainName "cohovines.com" `
-InstallDns:$true `
-LogPath "C:\Windows\NTDS" `
-NoRebootOnCompletion:$false `
-SiteName "Paris" `
-SysvolPath "C:\Windows\SYSVOL"`
-SafeModeAdministratorPassword ((Get-Credential).Password)
```

In this sample, you'll be prompted for the DSRM password. If you want to embed the password in your script, replace *(Get-Credential).Password* with *ConvertTo-SecureString -AsPlainText -Force -String "password"*.

If you want to demote a domain controller with PowerShell, you can use the *Uninstall-ADDSDomainController* cmdlet, as shown in the following sample:

```
Uninstall-ADDSDomainController -LocalAdministratorPassword `
(ConvertTo-SecureString -AsPlainText -Force -String "password") `
-Credential (Get-Credential)
```

The `-LocalAdministratorPassword` parameter specifies the local administrator password to set once the domain controller reboots as a member server. The `-Credential` parameter provides domain administrative credentials for the domain to complete the demotion.

If you're running an earlier version of Windows (prior to Windows Server 2012), automating the DC build process is more involved. You can make use of unattended answer files or the command-line interface for *dcpromo*. We generally recommend the answer file approach, as the command-line interface for *dcpromo* is not especially friendly. For details on the answer file format, review Microsoft Knowledge Base article 947034. For more information on using *dcpromo* from the command line, check out this link.

Virtualization

Server virtualization was never a consideration when Microsoft designed Active Directory, but over the course of Active Directory's evolution there have been various issues that have been identified as virtualization has come into play. The primary issue is that hardware virtualization typically offers the capability to roll a virtual machine back in time through the use of snapshots and similar technologies. When you consider the nature of tasks such as replication or issuing a relative identifier (RID) Active Directory expects that domain controllers should always be moving forward in time.

As these issues came to light, Microsoft made changes to Active Directory (starting in the Windows Server 2003 time frame) to detect scenarios where a snapshot might have been restored and in turn the domain controller's replication state rolled back. The term *USN rollback*, which describes this scenario, became well known among Active Directory administrators, and along with it came the fear of virtualizing domain controllers.

Fortunately, there is no reason domain controllers cannot be virtualized. There are best practices and security policies to take into account, but there's no reason that AD cannot run on a virtual hardware platform. In fact, Windows Server 2012 domain controllers can be notified of many rollback situations by the underlying virtualization host, and in turn they will insulate themselves from the rollback and get caught back up to the present state without any impact to the overall health of the environment.

When to Virtualize

There are a few risks that you should take into consideration when determining whether or not to virtualize some or all of the domain controllers in your forest. These risks revolve around security, redundancy, and the stability and operational practices of the host platform (hypervisor). Regardless of the version of Windows you're running Active Directory on, or the flavor of hypervisor, you will ultimately need to evaluate and plan in the same way.

First and foremost, it is extremely important that you ensure that your hypervisor of choice is listed in Microsoft's Server Virtualization Validation Program, or SVVP. Participation in the SVVP ensures that in the event you need to engage Microsoft support, you will be supported regardless of whether or not the server in question is virtualized or running on physical hardware.

When thinking about security, keep in mind the importance of physical security when discussing Active Directory domain controllers. Typically, you would think of whether or not the location where the domain controller is stored is secure (locked doors, video monitoring, etc.), but when a domain controller is virtualized, this question becomes more ambiguous. If the domain controller is virtualized, anyone who has access to the hypervisor storage where the virtual hard disks (VHDs) are stored effectively has physical access to the domain controller. Some organizations opt to operate a dedicated

virtualization infrastructure for Active Directory, while others accept this risk. Also keep in mind that many major hypervisors can ship virtual hard disk files to alternative locations over the network for disaster recovery and high availability purposes. This feature could be considered a positive or a negative; a virtual disk file you can copy for disaster recovery purposes can also be copied for more nefarious purposes. It is much easier to copy a file across the network than to physically remove a domain controller.

In large organizations, the team that runs Active Directory is often a small group of senior administrators who have years of experience and are very focused on delivering a narrow set of critical services (such as Active Directory). Simultaneously, large organizations often have separate teams that may not understand Active Directory that are responsible for operating the virtualization platform across the organization. In these situations, it is important to consider whether or not the team running the virtualization platform can consistently deliver the level of service required by the Active Directory team to maintain the service-level commitments for AD. You must also consider whether or not virtualization administrators can be trusted to follow special procedures for the treatment of domain controllers, such as not taking snapshots of the domain controller virtual machines. If these types of assurances are not possible, then it may be necessary to continue operating Active Directory on physical hardware.

Finally, factor in the level of redundancy that you will be able to achieve when virtualizing domain controllers. With physical hardware, it is relatively easy to ensure that multiple pieces of hardware are located properly within datacenters and networks to tolerate outages both at the domain controller level and other levels (such as power or network outages). When domain controllers are virtualized, it becomes much harder to understand these risks, given the layers of abstraction added by the virtualization host platform. At a minimum, you should ensure that multiple virtual domain controllers are configured such that they are hosted across multiple physical hosts and data storage devices within the environment. This is often referred to as *antiaffinity*. Also consider where restoring Active Directory lies in a datacenter or organization-level disaster recovery sequence. If you need to have Active Directory available before the virtualization host platform would be available, you will need to maintain physical domain controllers so that they can be brought online more rapidly.

Impact of Virtualization

Two key scenarios typically come into play when discussing the impacts of virtualization on a domain controller. The first is the concept of USN rollback. When we discussed virtualization in Chapter 6, we discussed the update sequence number (USN). The USN is what Active Directory uses to keep track of each change to the local database. The second scenario—RID reuse or duplication—is not as well known, but is still a risk. Any time a security principal is created, a unique relative identifier is issued to construct a unique security identifier (SID) for the security principal. Both of these scenarios

manifest themselves when unique numbers (USNs or RIDs) get reused due to a snapshot of a virtual machine being restored or rolled back.

USN rollback

USN rollback occurs because domain controllers keep track of the changes that they have replicated from another domain controller based on the highest USN corresponding to the changes that have replicated. Domain controllers do this in order to limit the number of objects that must be evaluated during each replication cycle, as well as to prevent unnecessary replication when changes may be received from multiple source domain controllers. These values are tracked in a table on each domain controller known as the *up-to-dateness* vector (UTDV). Every domain controller's database file (*ntds.dit*) is identified by a GUID called an *invocation ID*. Within the UTDV, domain controllers are identified by their invocation ID. For more information on the UTDV and invocation IDs, refer to Chapter 6.

Take the scenario in Figure 9-13. In this case, we have two domain controllers: DC-A and DC-B. DC-A is virtualized. Consider the events in the following timeline:

1. At T1, DC-A's `highestCommittedUSN` is `100` and its invocation ID is `{P}`. DC-B has an entry in its UTDV for DC-A that reflects this data. The domain controllers are both synchronized at this time.
2. At T2, the virtualization administrator takes a snapshot of DC-A.
3. Next, at T3, the Active Directory administrator creates 75 users on DC-A. DC-A's `highestCommittedUSN` is now 175 (as 75 changes have been made to the database).
4. At T4, the virtualization administrator reverts the snapshot of DC-A. DC-A is now at T2.
5. At T5, the Active Directory administrator creates 15 users on DC-A. DC-A's `highestCommittedUSN` is now 115. When DC-B replicates with DC-A, DC-B presents its UTDV to DC-A and DC-A sees that DC-B is in sync with DC-A through change 175. Thus, DC-A sends no changes to DC-B.

The scenario outlined here is exactly what comprises USN rollback. There is now an island of at least 15 changes on DC-A that will never replicate into the rest of the forest.

Active Directory has some protections built in for detecting USN rollback, starting with Windows Server 2003. These protections catch some but not all scenarios, thus making this an issue that can impact a forest. When AD does detect a rollback scenario, it isolates the domain controller by disabling inbound and outbound replication and pausing the NetLogon service. These actions aim to limit the damage to the environment and allow the administrator to demote the domain controller and repromote it. When this happens, AD will log an event (ID 2095, source NTDS Replication) similar to the following in the event log:

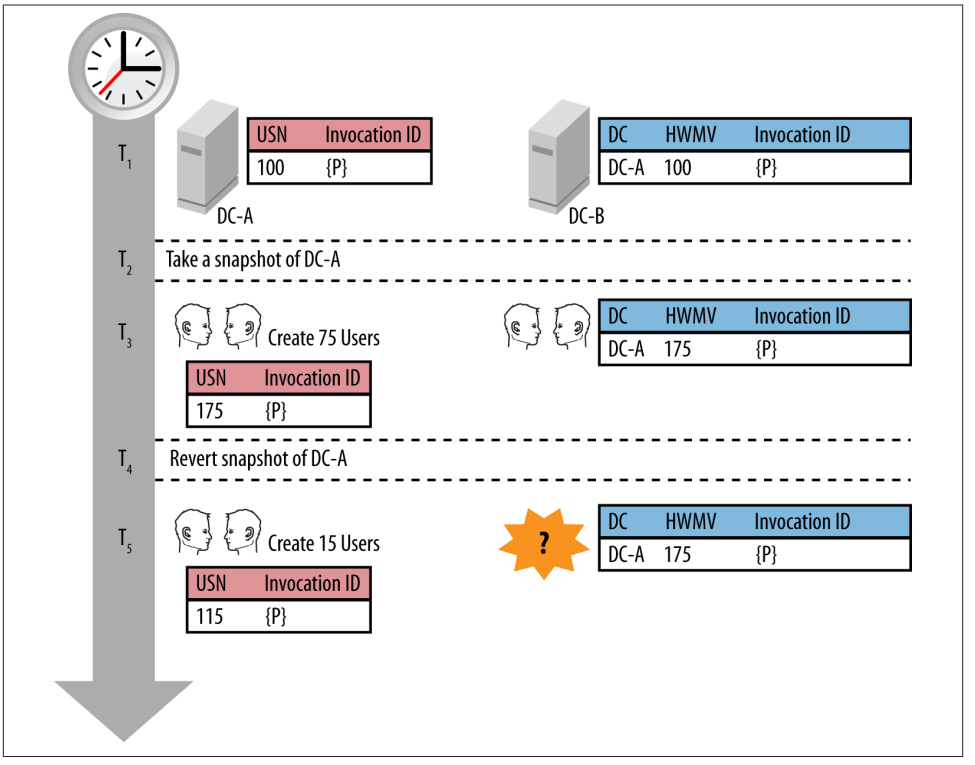


Figure 9-13. USN rollback timeline

During an Active Directory Domain Services replication request, the local domain controller (DC) identified a remote DC which has received replication data from the local DC using already-acknowledged USN tracking numbers.

Because the remote DC believes it is has a more up-to-date Active Directory Domain Services database than the local DC, the remote DC will not apply future changes to its copy of the Active Directory Domain Services database or replicate them to its direct and transitive replication partners that originate from this local DC.

If not resolved immediately, this scenario will result in inconsistencies in the Active Directory Domain Services databases of this source DC and one or more direct and transitive replication partners. Specifically the consistency of users, computers and trust relationships, their passwords, security groups, security group memberships and other Active Directory Domain Services configuration data may vary, affecting the ability to log on, find objects of interest and perform other critical operations.

To determine if this misconfiguration exists, query this event ID using *http://support.microsoft.com* or contact your Microsoft product support.

The most probable cause of this situation is the improper restore of Active Directory Domain Services on the local domain controller.

User Actions:

If this situation occurred because of an improper or unintended restore, forcibly demote the DC.

Remote DC: a2c53d4a-0018-4641-b277-eaecd25ec279

Partition: DC=contoso, DC=com

USN reported by Remote DC: 1000

USN reported by Local DC: 900

For more information on the USN rollback detection mechanism in AD, refer to Microsoft Knowledge Base article 875495.

RID pool reuse

Similar to the scenario illustrated in Figure 9-13 is the impact of reusing RIDs. As we discussed in Chapter 2, RIDs are unique numbers that are used to construct a unique SID for each security principal (user, group, or computer) in the domain. Once a RID is used, it can never be reused. If you consider that at T3 in Figure 9-13, RIDs 100–175 were used, when the domain controller is rolled back and users are created at T5, RIDs 100–115 will be reused. This presents a security challenge since multiple users have the same unique identifier that corresponds to access control lists across the forest. To mitigate this problem, when Active Directory encounters more than one object with the same RID, it immediately deletes both objects from the database.

System clock changes

Virtual machines are dependent on the host platform to provide accurate time information during boot up, at a minimum. By default, most virtualization platforms also configure guests to synchronize their clocks with the underlying host's clock. Both of these scenarios lead to challenges when the virtualization host's clock is not accurate.

Active Directory includes the current time in a number of places. These locations include replication metadata as well as attribute-level timestamps such as `lastLogonTimeStamp`. If the DC's clock jumps forward, invalid timestamps will start appearing in Active Directory. In the case of `lastLogonTimeStamp`, the invalid future timestamp will remain in place until it is superseded. If your clock jumps forward years at a time, for example, you will have users and computers with invalid `lastLogonTimeStamp` values, among other things.

Virtualization Safe Restore

As virtualization became iteratively more mainstream, it became clear that Active Directory could not be a product facing significant challenges in common virtualization scenarios. To aid with virtualization, Windows Server 2012 domain controllers support the notion of a *virtual machine generation identifier* (VM Gen ID). The VM Gen ID is

a unique identifier for the virtual machine's state that is projected into the virtual machine by the underlying virtualization host platform.

At the time of writing, only Microsoft's Hyper-V platform supports VM Gen IDs, but we expect that this will expand to other major virtualization host platforms over time.

The goal of the VM Gen ID is to detect scenarios where the domain controller could be reverted in time, such as by applying a snapshot or copying a VHD file. In general, the underlying principle of when a VM Gen ID is updated relates to when a time interval will reoccur. If you think of Active Directory's USN as a logical clock—a number that always ticks forward—then any scenario at the virtualization level where the guest's USN could tick backward would require the VM Gen ID to be updated. Any time Active Directory detects the VM Gen ID has rolled back, AD will report the following event:

> Event ID: 2170
>
> Task Category: Internal Configuration
>
> Level: Warning
>
> Description:
>
> A Generation ID change has been detected.
>
> Generation ID cached in DS (old value): 4614786344935980275
>
> Generation ID currently in VM (new value): 10595675270371099562
>
> The Generation ID change occurs after the application of a virtual machine snapshot, after a virtual machine import operation or after a live migration operation. Active Directory Domain Services will create a new invocation ID to recover the domain controller. Virtualized domain controllers should not be restored using virtual machine snapshots. The supported method to restore or rollback the content of an Active Directory Domain Services database is to restore a system state backup made with an Active Directory Domain Services aware backup application.

Active Directory keeps the current VM Gen ID in memory (and in a nonreplicated attribute in the AD database, `msDS-GenerationID`) and compares that value to the current VM Gen ID reported by the virtualization host platform prior to a change being committed to the Active Directory database, as well as during boot up. If the VM Gen ID has changed, the domain controller first takes the following steps:

1. Reset the domain controller's invocation ID.
2. Release the domain controller's RID pool.
3. Nonauthoritatively restore the Sysvol share on the domain controller.

4. If the domain controller is a FSMO role holder, Active Directory will also pause handling of the functions related to the FSMO roles until a replication cycle can be completed.

These steps collectively ensure that the domain controller will become consistent with the forest as a whole and that there will be no bubbles in time, duplication, or data that is not consistent across the forest.

Cloning Domain Controllers

One of the interesting new scenarios that the virtualization safe restore functionality introduces is the ability to rapidly clone virtual domain controllers. This scenario enables you to rapidly scale out to meet demand (such as in a cloud scenario), to deploy new virtual domain controllers (perhaps as part of an upgrade project), or to rapidly deploy domain controllers in a disaster recovery scenario. Fundamentally, you can duplicate a Windows Server 2012 domain controller's virtual hard disk file and create as many additional unique domain controllers as you need to.

In order to leverage this functionality, there are two prerequisites. The first is that your domain's PDC emulator FSMO role holder is running Windows Server 2012. The second prerequisite is that the virtualization host platform supports the VM Gen ID functionality. Once you have met these two prerequisites, you can start creating clones.

When you clone a domain controller, Active Directory needs a way to know that you're creating a clone rather than rolling a snapshot back, for example. To signify this to Active Directory, you need to create a *cloning configuration file* that has the cloning configuration and make it available to the cloned VHD at boot up. The easiest way to create a cloning configuration file is with the *New-ADDCCloneConfigFile* PowerShell cmdlet.

The cloning configuration file contains unique identifying elements that will be applied to the clone, such as computer name, IP address, and other network information. Any information not supplied in the configuration file will be dynamically generated during the first boot of the clone. For example, if IP information is not specified, DHCP will be used. If the computer name is not specified, a name will be dynamically generated.

The format for a dynamically generated computer name is the first eight characters of the source DC's name followed by *–CL####*. The final four numbers are from 0001 to 9999. During cloning authorization, the PDC emulator will pick an available number to uniquely name the clone.

If you provide a blank configuration file like this, then AD will dynamically generate all of the necessary parameters at startup:

```
<?xml version="1.0"?>
<d3c:DCCloneConfig xmlns:d3c="uri:microsoft.com:schemas:DCCloneConfig">
</d3c:DCCloneConfig>
```

Table 9-1 shows the parameters that you can specify to *New-ADDCCloneConfigFile*.

Table 9-1. Cloning configuration file parameters

Name	Description
`Path`	The path indicating where to create the *DCCloneConfig.xml* file. If you don't specify this parameter, it will be created in the *DSA Working Directory* folder (generally where the *ntds.dit* file is stored).
`CloneComputerName`	The name of the cloned domain controller.
`SiteName`	The AD site to place the cloned domain controller in.
`IPv4Address`	The static IPv4 address to assign to the clone.
`IPv4SubnetMask`	The static IPv4 subnet mask to assign to the clone.
`IPv4DefaultGateway`	The static IPv4 default gateway to assign to the clone.
`IPv4DNSResolver`	The IPv4 DNS server(s) to assign to the clone. Use array syntax to specify more than one DNS server—for example: `-IPv4DNSResolver @("10.10.10.1", "10.10.10.2")`
`PreferredWINSServer`	The primary WINS server to assign to the clone.
`AlternateWINSServer`	The secondary WINS server to assign to the clone.
`IPv6DNSResolver`	The IPv6 DNS server(s) to assign to the clone. Use array syntax, as shown in the `IPv4DNSResolver` sample, to specify more than one.
`Static`	If you are specifying static IP address or DNS server information, you must include this switch.
`Offline`	Include this switch to bypass validation checks before creating the cloning configuration file.

You cannot specify a static IPv6 address in the DC cloning configuration file.

There are three locations where you can place the DC cloning configuration file (*DCCloneConfig.xml*). Which location you choose will depend on your use of the cloning feature as well as your deployment strategy. These locations, in order, are:

- The *DSA Working Directory* folder (where the *ntds.dit* file is stored)
- The *%windir%\ntds* folder
- The root of some mounted removable media (such as a virtual floppy disk), in alphabetical order of the drive letter

If you will be cloning DCs on an ad hoc basis or if you want to simply use a blank or generic cloning configuration file, chances are you'll want to use either the first or second option listed. If you'll be cloning numerous DCs from the same source and you want to have a specific configuration for each DC, you'll probably want to look at the third option. This will allow you to create custom virtual floppy disk (VFD) files for each clone DC that contains the clone's specific configuration settings in the *DCCloneConfig.xml* file and then mount that VFD file during the first boot. Alternately, you could mount the cloned VHD file prior to the first boot (with the *Mount-VHD* Hyper-V PowerShell cmdlet, for example) and inject the *DCCloneConfig.xml* file into locations one or two.

You can also manually mount and unmount a VHD file from Windows Explorer. Select the VHD file and then click "Mount" on the Disc Image Tools contextual ribbon tab, as shown in Figure 9-14. Once you are done, select the mounted drive and click Eject on the ribbon.

In addition to the *DCCloneConfig.xml* file, Microsoft has added another important safeguard—a *cloning allow list*. One of the risks of implementing cloning that Microsoft identified is that applications or services (such as backup agents or antivirus software) on the DC to be cloned might not support cloning. To mitigate this, you must explicitly tell Active Directory that a DC with applications and services installed outside of the basics of Windows can be cloned by specifying those applications and services in the *CustomDCCloneAllowList.xml* file.

You're probably wondering about the impacts of cloning on Windows itself. When a DC is cloned, during the first boot, it will run an abbreviated version of the *sysprep* process that is used when cloning a traditional server or workstation in order to remove identifying details specific to the source DC.

You can find a list of *sysprep* providers that are run as well as the default allowed applications and services list by reviewing the file *DefaultDCCloneAllowList.xml* in *%windir%\system32*.

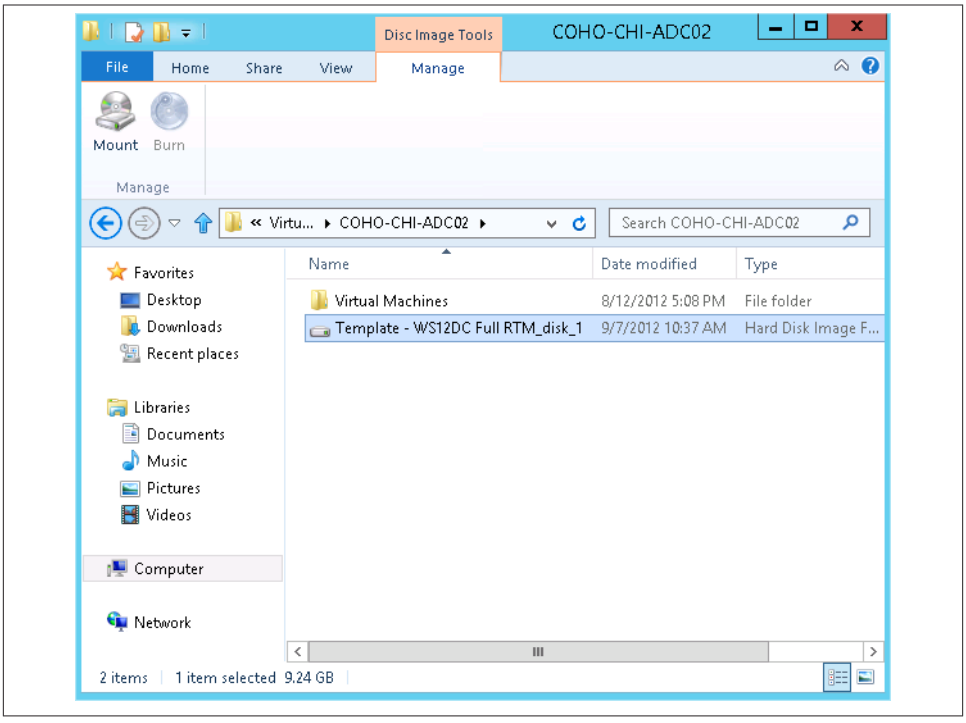


Figure 9-14. Mounting a VHD with Windows Explorer

First, run *Get-ADDCCloningExcludedApplicationList* on the domain controller that you will be cloning. This PowerShell cmdlet will check all of the services against an approved list, as well as all of the installed applications. If any custom applications are found, they will be printed to the PowerShell window. Once you have removed any applications that do not support cloning from the source DC, rerun *Get-ADDCCloningExcludedApplicationList* with the *-GenerateXml* switch and the *-Path* argument. PowerShell will create an allow list XML file.

While it is possible to edit both the *DCCloneConfig.xml* and *CustomDCCloneAllowList.xml* files by hand, you will need to be extremely careful that you do not invalidate the XML. If you do, the cloning process will fail.

The final step is that you must add the source DC's computer account to the Cloneable Domain Controllers group in Active Directory. This grants the source DC the rights to clone itself.

The DC cloning process

Once you've completed the prerequisite steps, duplicated the virtual hard disk file, created the configuration files, and started the cloned virtual machine for the first time, Active Directory's cloning process will begin. There are a number of steps involved in this process. Many steps have fail-safe processes that ensure that in the event of a failure, the domain controller reboots into Directory Services Restore Mode (DSRM) so that the failure will not negatively impact the environment.

During boot, Active Directory will first check if the VM Gen ID has changed. If the VM Gen ID has changed, then Active Directory knows that something has happened. Based on the presence of the *DCCloneConfig.xml* file, Active Directory knows that it needs to begin the cloning process.

If Active Directory finds that the VM Gen ID is not present, but a *DCCloneConfig.xml* file is present, the domain controller renames the *DCCloneConfig.xml* file and reboots into DS Restore Mode. This ensures that a clone is not introduced into the network that is not running a VM Gen ID-aware virtualization host platform.

First, Active Directory completes all of the steps described in the section "Virtualization Safe Restore" on page 220. Next, the cloning process begins. Active Directory takes the following steps to complete the cloning process:

1. The server processes the *DCCloneConfig.xml* file. If a static IP specified in the configuration file will cause an IP address conflict, the clone boots into DS Restore Mode.
2. DNS registration is disabled on all NICs and the NetLogon service is stopped to ensure that the clone will not be contacted.
3. The list of installed applications and services is validated against the built-in allow list as well as the *CustomDCCloneAllowList.xml* file. If there are applications or services that are not listed, the clone boots into DS Restore Mode.
4. The PDC emulator FSMO is contacted via RPC to begin the cloning process.
5. The PDC emulator ensures that the source VM for the clone is permitted to clone itself. If the necessary permissions are not in place, the clone boots into DS Restore Mode.
6. The clone's computer name is changed and then the necessary server and NTDS Settings objects are created in the clone's Active Directory site.
7. The clone promotes itself as a domain controller using the Install from Media promotion process. This enables the clone to be promoted without replicating anything other than changes to AD or Sysvol since the virtual hard disk was cloned.

8. The clone runs *sysprep* and reboots itself. A registry flag is set in `HKLM\System\CurrentControlSet\Services\NTDS\Parameters\VdcCloningDone` to indicate that the DC is a cloned domain controller.

If the source DC that is being cloned was the PDC emulator, there is an extra reboot in between steps six and seven. While the preceding list provides a high-level overview of the key steps involved in a cloned domain controller coming online, there is a far more extensive description of all of the details available.

Cloning a domain controller

To wrap up our discussion of cloning, we'll clone a DC called *COHO-SFO-ADC01* and create a new DC called *COHO-PAR-ADC01*. To do this, we'll duplicate *COHO-SFO-ADC01*'s virtual hard disk file, mount the virtual hard disk, and then inject the *DCCloneConfig.xml* file. Finally, we'll boot the new VM (*COHO-PAR-ADC01*) and observe the cloning process as it proceeds. Our clone will have the properties shown in Table 9-2.

Table 9-2. Cloned domain controller properties

Property	Value
Name	COHO-PAR-ADC01
Active Directory Site	Paris
IPv4 Address	172.16.4.1
IPv4 Subnet Mask	255.255.255.0
IPv4 Default Gateway	172.16.4.1
IPv4 DNS Servers	172.16.1.21 172.16.1.22

Here are the steps you'll take to accomplish this:

1. On *COHO-SFO-ADC01*, run *Get-ADDCCloningExcludedApplicationList*. If any applications that do not support cloning are listed, uninstall them.
2. On *COHO-SFO-ADC01*, run *Get-ADDCCloningExcludedApplicationList -GenerateXML* to generate a custom cloning allow list.
3. Add *COHO-SFO-ADC01* to the Cloneable Domain Controllers group. You can do this with PowerShell by running this command:

```
Add-ADGroupMember "Cloneable Domain Controllers" -Members COHO-SFO-ADC01$
```

4. Shut down *COHO-SFO-ADC01* and duplicate the VM to create a VM for *COHO-PAR-ADC01*.
5. Create a cloning configuration file for *COHO-PAR-ADC01*. You can do this on any machine with the Active Directory RSAT tools installed by running the following PowerShell command:

```
New-ADDCCloneConfigFile -CloneComputerName COHO-PAR-ADC01 `
-SiteName Paris `
-IPv4Address 172.16.4.21 `
-IPv4SubnetMask 255.255.255.0 `
-IPv4DefaultGateway 172.16.4.1 `
-IPv4DNSResolver @("172.16.1.21", "172.16.1.22") `
-Static `
-Offline `
-Path .\
```

6. Mount the virtual hard disk file and copy the new *DCCloneConfig.xml* file to *%windir%\ntds* in the mounted virtual hard disk. To do this with Hyper-V, follow these steps:

```
Mount-VHD -Path `
 'E:\Hyper-V\Virtual Machines\COHO-PAR-ADC01\COHO-PAR-ADC01.vhdx'
Dismount-VHD -Path `
 'E:\Hyper-V\Virtual Machines\COHO-PAR-ADC01\COHO-PAR-ADC01.vhdx'
```

7. Boot *COHO-PAR-ADC01*. If you watch the boot process, you will see the cloning take place, similar to what's shown in Figure 9-15 and Figure 9-16.

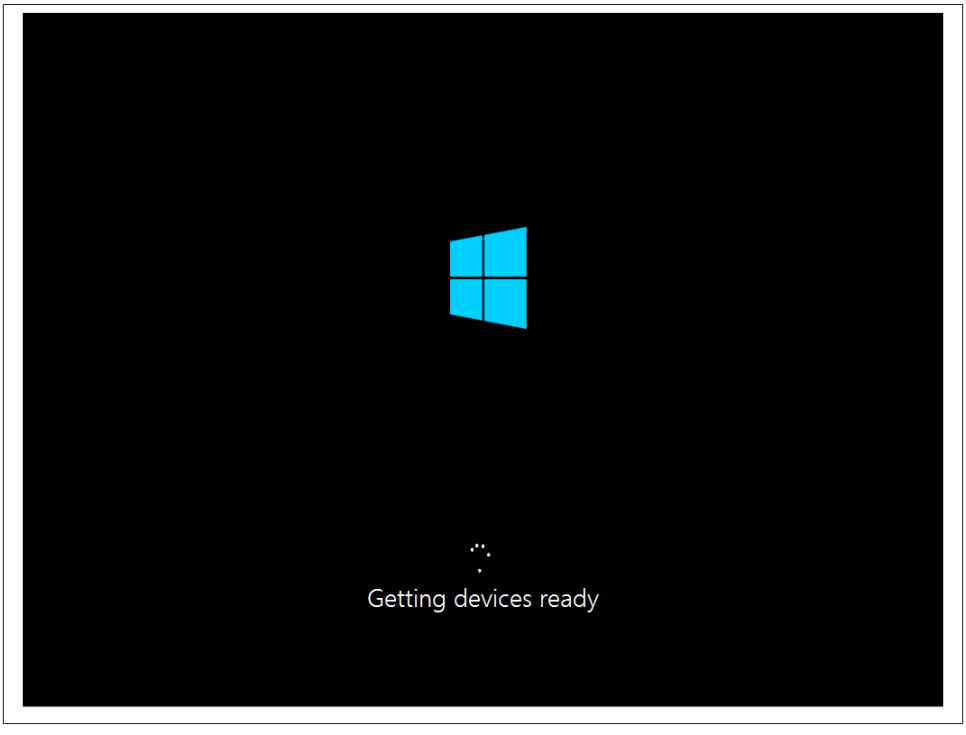


Figure 9-15. Abbreviated sysprep during domain controller cloning

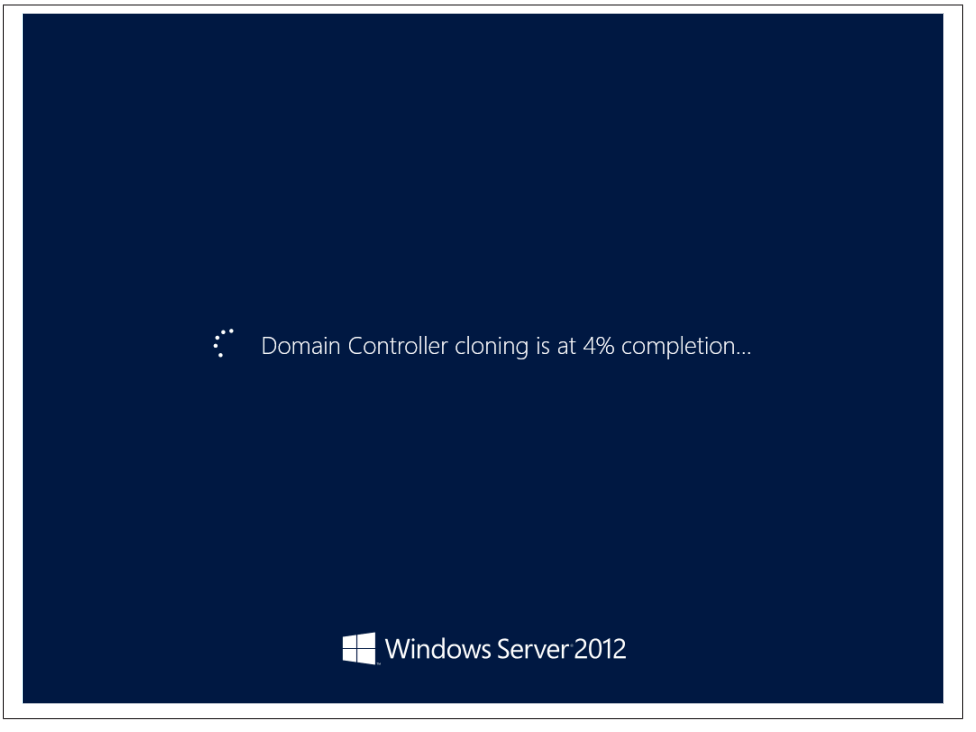


Figure 9-16. Domain controller cloning in progress

Once the boot process completes, you will have a fully functional cloned domain controller that is ready to service client requests.

Read-Only Domain Controllers

One of the most significant Active Directory features introduced in Windows Server 2008 was the read-only domain controller. Deploying domain controllers into untrusted locations has always been a substantial security risk for Active Directory deployments. The risk of a domain controller being physically compromised and having password hashes for that domain stolen and the risk of the database (*ntds.dit*) being modified offline and placed back on the network are both important ones to consider. The RODC brings the ability to mitigate both of these risks.

By default, RODCs do not store any passwords locally in their database. If a user authenticates to an RODC, the RODC will need to contact a writable domain controller (sometimes called an *RWDC*) upstream in order to validate that user's password. This, of course, also applies to other objects with passwords, such as computer and trust accounts. Through the use of password replication policies discussed later in this

section, you can define what passwords are allowed to be cached locally on an RODC. You can also examine a real-time view of what passwords are currently cached on an RODC.

In order to ensure that an RODC cannot impact the integrity of an Active Directory forest, all replication to RODCs is one-way. This means that if someone manages to make a change on an RODC, Active Directory will not replicate that change to other Domain Controllers. Figure 9-17 shows the replication paths in a network with RODCs deployed.

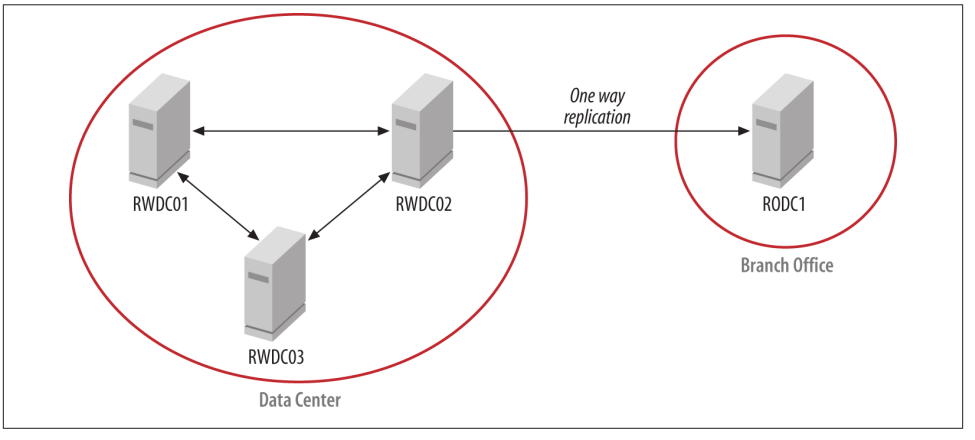


Figure 9-17. RODC replication

Generally speaking, there are a couple of scenarios where we see RODCs being deployed. The first is in the branch office. Branch offices are typically smaller facilities that are connected by a WAN link to a datacenter or other central site. These WAN links vary greatly in speed and reliability. The second key characteristic of the branch office is the inability for the physical security of a domain controller to be guaranteed. Oftentimes, server rooms in branch offices are merely closets or other repurposed spaces that have limited provisions for managing who has access to the space. The majority of our discussion about RODCs in this book will be focused on the branch office.

The second scenario is the concept of the RODC in the perimeter network. The *perimeter network* is a section of a network that is considered untrusted relative to the rest of the network. Many organizations place their servers that are Internet-facing in a perimeter network. In order for these servers to be joined to the domain, numerous ports must be opened through the firewall, to the point that the firewall is often (accurately) compared to Swiss cheese. Some organizations may elect to deploy RODCs into their perimeter network to serve as domain controllers for the servers alongside them in the perimeter network. Using an RODC allows local network authentication while mitigating the risk of someone compromising the domain controller and making changes

that replicate back to the domain controllers on the internal network. You still run the risk of internal information disclosure, however (e.g., phone numbers, corporate reporting structure, and any other internal information).

As you read this section, keep in mind the guiding design principle behind the RODC. That principle is the core assumption that the RODC is compromised by default. If you're wondering why a feature of the RODC behaves in a certain way, take a moment to consider the scenario where the RODC is compromised.

Prerequisites

There are a few major prerequisites to deploying RODCs into your Active Directory installation that you should take into consideration before proceeding any further. The first is that your forest must at a minimum be at the Windows Server 2003 forest functional level. Next, you'll need to have a writable Windows Server 2008 domain controller in a site that your RODC is connected to, as determined by the Active Directory site-link topology.

Consider Figure 9-18. If you have "Bridge all site links" enabled in Active Directory Sites and Services, RODCs can be more than one hop away from a Windows Server 2008 or newer RWDC, and thus the RODC in Site A will be able to communicate with the Windows Server 2008 RWDC in Site C. If, however, you do not have "Bridge all site links" enabled, you will either need to create a site link bridge that includes site links *A - B* and *B - C*, or deploy a Windows Server 2008 or newer RWDC in Site B.

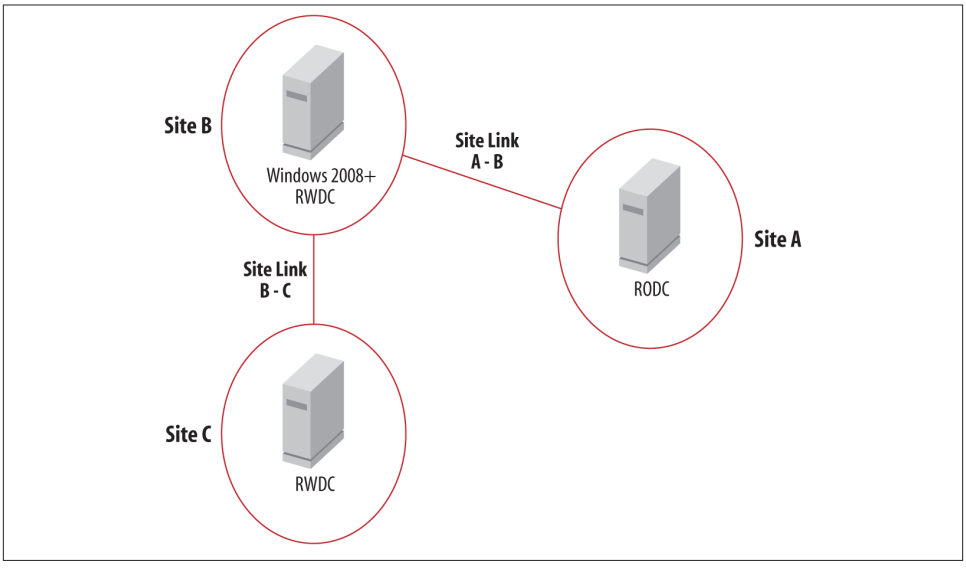


Figure 9-18. Sample RODC deployment topology

In addition to these Active Directory requirements, you should investigate deploying the RODC compatibility pack to your clients. This package is available for download via (and described in) Microsoft Knowledge Base article 944043. There are a number of compatibility issues and bugs with Windows XP and Windows Server 2003 clients and domain controllers that you are likely to encounter, and this package solves many of them. We will make reference to situations where this package may be required throughout this chapter when appropriate.

In this chapter, we will often reference situations where an RODC can request replication of a specific object. This functionality is known as *Replicatesingleobject* (RSO). The replicate-single-object operation is initiated via a modification to the `replicateSingleObject` operational attribute of the `RootDSE` LDAP entry. For more information on this attribute, reference this site.

Password Replication Policies

As we discussed briefly, RODCs don't cache passwords by default. This means that if an attacker were to get a copy of the Active Directory database from an RODC, she would not be able to compromise any passwords since there are none stored there. The downside of not storing any passwords is that the RODC will need to make a call to an RWDC, usually over the WAN, to authenticate a client. The value of the RODC diminishes when the WAN is down and clients cannot be authenticated even though there is a local domain controller.

The solution to this problem is the use of *password replication policies* (PRPs). Password replication policies allow you to define what user (and computer) passwords are cached locally on the RODC, as well as which passwords can never be cached. PRPs are defined through a series of four linked multivalued attributes on the RODC's computer account in Active Directory. Those attributes are:

`msDS-RevealOnDemandGroup`
: This is the list of principals for which the RODC can cache the password. This list is often referred to as the *allowed* list.

`msDS-NeverRevealGroup`
: This is the list of principals for which the RODC is *never* allowed to cache the password. This list is often referred to as the *denied* list.

`msDS-RevealedList`
: This is a list of principals whose passwords are currently cached on the RODC. It is often referred to as the revealed list.

`msDS-AuthenticatedAtDC`
: This is a list of principals who have authenticated to the RODC. This list is often referred to as the authenticated to or *Auth-2* list.

In addition to these attributes, there are two new built-in groups that are added to Active Directory when you introduce the first RODC in the domain. These are the *Allowed RODC Password Replication Group* and the *Denied RODC Password Replication Group*. The allowed group is added to the `msDS-RevealOnDemandGroup` attribute by default, and the denied group is added to the `msDS-NeverRevealGroup` attribute by default. The allowed group has no members by default but the denied group has several:

- Cert Publishers
- Domain Admins
- Enterprise Admins
- Enterprise Domain Controllers
- Enterprise Read-Only Domain Controllers
- Schema Admins
- *krbtgt* account

In addition to these groups and accounts, there are a few groups that are added to the `msDS-NeverRevealGroup` attribute by default:

- Account Operators
- Administrators
- Backup Operators
- Denied RODC Password Replication Group
- Server Operators

All of these groups generally contain highly trusted user accounts, so you would not want to risk compromising those passwords by caching them on an RODC. Thus, we don't recommend that you remove these groups from the default denied replication group, or the `msDS-NeverRevealGroup` attribute.

In order for an RODC to issue a Kerberos service ticket without communicating with a writable domain controller, the RODC will need to have the passwords cached locally for both the user account and the computer account involved. This is important to remember, as you will need to plan for both user and computer account password caching. In addition to your users, you need to take into consideration any service accounts that are being used on servers or other machines at the site where the RODC is located.

As you plan your RODC deployment, you'll need to think carefully about how you set up your password replication policies. Depending on the route you take, you may need to invest in updating your user provisioning process to keep the password replication policies updated. There are three strategies we expect most organizations will take when planning their password replication policies:

No caching of passwords
: In this scenario, RODCs will not cache any passwords and will instead simply serve as read-only directories that are proxies in the authentication process. If you are considering deploying an RODC in the perimeter network scenario, this is likely the route you will take when defining password replication policies for those RODCs. This is the most secure configuration and is the default. No management of policies or groups are required.

Caching of almost all passwords
: If you are looking to reduce management overhead and provide the most tolerance for WAN failures, this is likely the scenario for your password replication policies. If you have physical security concerns about an RODC, we do not recommend that you opt for this strategy. Passwords will only be cached on a given RODC once the user or computer has logged on via that specific RODC or the password is manually replicated to the RODC.

Limited caching of passwords
: In this scenario, you will define on a per-RODC basis which passwords can be cached by the RODC. This list will generally reflect the list of users and computers that are located at the site the RODC is servicing. You will need to devise a process to manage the lists for each of your RODCs in this situation. We recommend that you integrate this process with your user-provisioning process. In addition to your user-provisioning process, you will need to link into a mechanism that manages your computer accounts. Don't forget to consider users who move between sites when you plan for this scenario.

Managing the password replication policy

The Active Directory Users and Computers (ADUC) tool includes a property page for managing an RODC's PRP, as well as for inspecting the revealed and authenticated to lists. Figure 9-19 shows the default settings for a new RODC. The list shown here is accessible via right-clicking the domain controller's computer account in ADUC and selecting Properties.

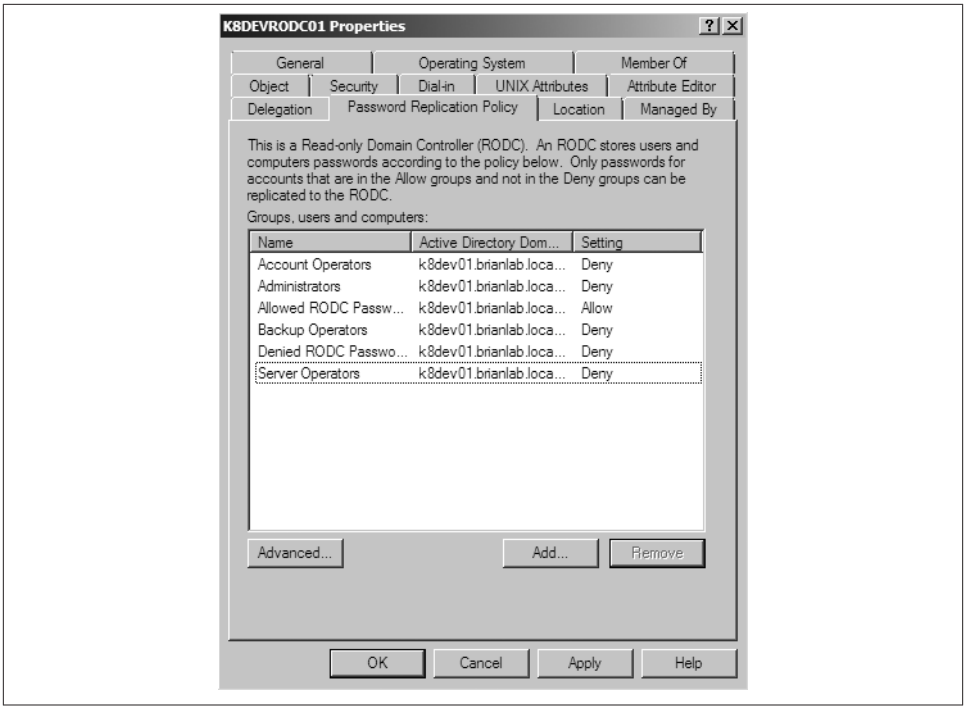


Figure 9-19. ADUC Password Replication Policy tab

You can add users, groups, and computers to the list here by clicking Add. When you click Add, ADUC will prompt you to allow or deny replication for the selection, as shown in Figure 9-20.

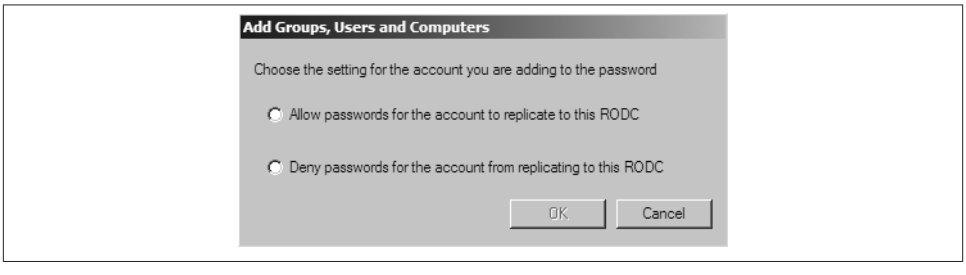


Figure 9-20. Adding to the password replication policy lists

Clicking the Advanced button in Figure 9-19 yields a dialog that allows you to inspect the `msDS-RevealedList` and `msDS-AuthenticatedAtDC` attributes. The drop-down list at the top of Figure 9-21 allows you to toggle between which attributes you want to display. "Accounts whose passwords are stored on this Read-only Domain Controller"

displays the `msDS-RevealedList` attribute, and "Accounts that have been authenticated to this Read-only Domain Controller" displays the `msDS-AuthenticatedAtDC` attribute. If you click the Export button at the lower-left of the dialog shown in Figure 9-21, you can save a CSV (comma-separated values) listing of the data shown.

The Advanced dialog shown in Figure 9-21 also allows you to determine what accounts can have their passwords cached by an RODC. This is a useful feature when you're troubleshooting an account whose password caching behavior is unexpected. Figure 9-22 shows the Resultant Policy tab for a number of users.

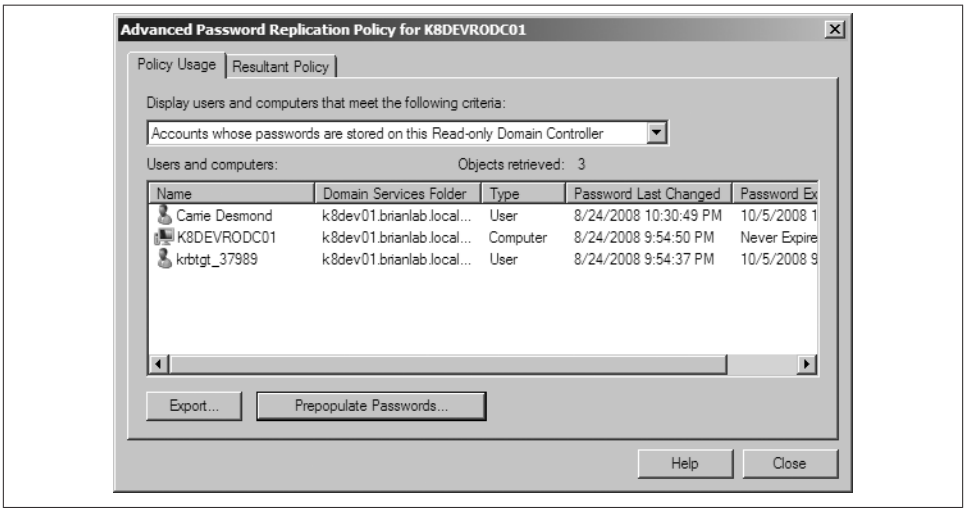


Figure 9-21. Viewing the revealed list

There are a few different resultant policy settings that can be displayed for an account. An account that shows "Allow" is defined on the `msDS-RevealOnDemandGroup` list. An account that shows "Deny (explicit)" is defined on the `msDS-NeverRevealGroup` list. Finally, an account showing "Deny (implicit)" is not defined on either list and thus is not allowed to cache its password, since the default behavior for an RODC is not to cache a given account's password. Notice that, as shown in Figure 9-22, you can calculate resultant policies for both users and computers.

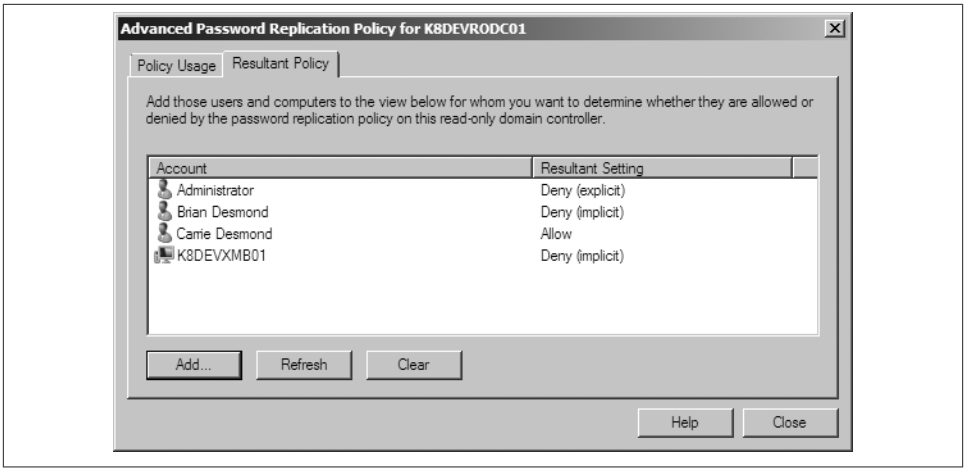
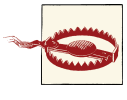


Figure 9-22. Resultant password replication policy

Managing the loss of an RODC

In the event that an RODC is stolen—or, more specifically, the drives holding the database are physically compromised or stolen—you should as a matter of best practice immediately reset the passwords that were cached on that RODC.

The ADUC user interface for performing this task is accessed by deleting the RODC's computer account from the Domain Controllers OU. If you attempt to delete the computer account, you'll be presented with the dialog shown in Figure 9-23. From this dialog, you can elect to reset the passwords for users, computers, or both. While the dialog only resets user passwords by default, keep in mind that computers are a special type of user account and they have passwords, too. An attacker could use a computer's account and password to authenticate to your domain.



Be certain that you heed the warnings in Figure 9-23. Not resetting passwords is a security risk, so we don't recommend that you elect not to reset passwords in the name of convenience. We do, however, recommend that you pause to consider the implications of resetting all of the cached user passwords in a site. This will cause service desk calls for users who can't log in and, in the case of service accounts, services that can't start.

When you reset computer passwords, the computers will be unable to authenticate to the domain. You will need to physically visit each workstation or script a solution using tools like *netdom* or *nltest* if you have the local administrator credentials for each machine available centrally.

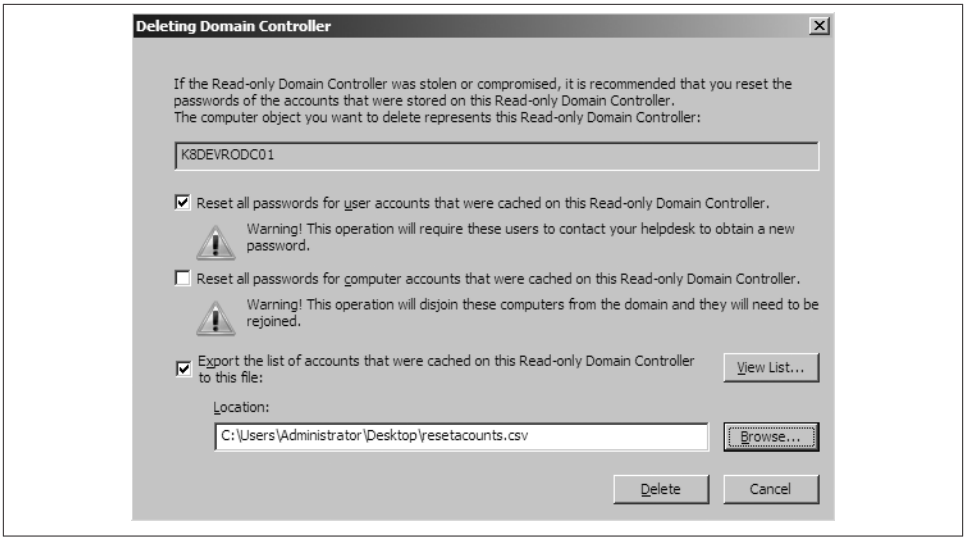


Figure 9-23. Deleting an RODC computer account

While it's not required, we highly recommend that you export a CSV list of the affected accounts so that you can follow up with your service desk and other support teams. Resetting user and computer passwords will undoubtedly cause substantial pain, so this is a scenario you should plan for as you plan your overall RODC deployment.

The Client Logon Process

So far we've discussed the caching of passwords on RODCs and how to control the caching, but we have not looked at how the RODC actually gets passwords from a writable domain controller and caches them locally. In this section, we'll take a look not only at how passwords get cached locally, but also at the overall flow of communications between domain controllers and clients when an RODC is involved.

One key difference between RODCs and their writable counterparts that you should keep in mind throughout this section (and that we will frequently point out) is that each RODC has its own *krbtgt* account. The domain *krbtgt* account is the credential that is used by domain controllers to encrypt Kerberos tickets, and if you have access to these credentials, you have an endless degree of power over the forest. Consequently, each RODC maintains a separate *krbtgt* account that is named in the format *krbtgt_<number>*, where *<number>* is a unique number. The RODC stores only the password for its personal *krbtgt* account locally. Writable domain controllers, however, store the passwords for each *krbtgt* account in their databases.

In order to determine what *krbtgt* account is associated with a given RODC, you can inspect the `msDS-KrbTgtLink` attribute on an RODC's computer account, or the `msDS-krbTgtLinkBL` attribute on the *krbtgt* account.

Let's consider the situation depicted in Figure 9-24 as our baseline for the rest of this discussion. Figure 9-24 has four players:

RWDC01
: This is our writable domain controller in the datacenter.

RODC01
: This is our read-only domain controller in the branch office.

User Brian
: This is our user who will be logging on and accessing services.

Machine PC01
: This is the workstation for user *Brian*.

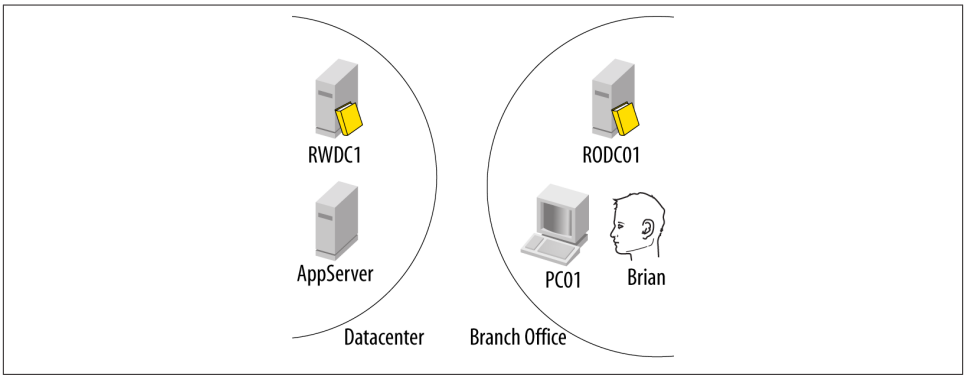


Figure 9-24. Sample network topology

Let's assume that the first step is for our machine, *PC01*, to authenticate to the domain. The following are the steps that will take place:

1. First, *PC01* will contact *RODC01* and provide a Kerberos authentication service request (a *KRB_AS_REQ* packet).
2. When *RODC01* receives the *KRB_AS_REQ*, it will check its local database to see if it already has *PC01*'s password cached locally. For this example, we'll assume that this is, in fact, not the case.
3. Since *RODC01* does not have the password cached locally, it must contact *RWDC01* and provide the *KRB_AS_REQ* it received.

4. *RWDC01* will generate a *KRB_AS_REP* (a Kerberos authentication service reply) and forward it back to *RODC01*.
5. *RODC01* will in turn provide this reply to *PC01*, completing the authentication request. *RWDC01* will also update *RODC01*'s Auth-2 list (`msDS-AuthenticatedAtDC`) to include *PC01* in that list.

   At this point, *PC01* has a valid Kerberos ticket-granting ticket (TGT) signed with the domain *krbtgt* account. Following step 5, *RODC01* will initiate two additional steps to attempt to cache the password locally:
6. *RODC01* will submit a request to *RWDC01* to have *PC01*'s credentials replicated to *RODC01*.
7. *RWDC01* will verify that the password replication policy permits *RODC01* to cache *PC01*'s password and replicate the password to *RODC01*. *RWDC01* will also update *RODC01*'s revealed list (`msDS-RevealedList`) to include *PC01*.

In looking at step 7, you may be wondering why *RWDC01* checks the password replication policy rather than assuming *RODC01* can cache the password since it requested it. The reason here is that we must assume that *RODC01* has been compromised, and that as part of that incident the local copy of the password replication policy on *RODC01* may have been modified and additional accounts surreptitiously added to the PRP allowed list.

Let's now look at the process of user *Brian* logging in to his machine. We'll of course assume that the preceding seven steps have completed successfully. The following are the steps taken when Brian attempts to log into his machine:

1. Brian presents his credentials to *PC01*, and *PC01* sends a *KRB_AS_REQ* to *RODC01* for user *Brian*.
2. When *RODC01* receives the *KRB_AS_REQ*, it will check its local database to see if it already has Brian's password cached locally. For this example, we'll assume that this is, in fact, not the case.
3. Since *RODC01* does not have the password cached locally, it must contact *RWDC01* and provide the *KRB_AS_REQ* it received.
4. *RWDC01* will generate a *KRB_AS_REP* and forward it back to *RODC01*.
5. *RODC01* will in turn provide this reply to *PC01*, completing the authentication request for user *Brian*. *RWDC01* will also update *RODC01*'s Auth-2 list (`msDS-AuthenticatedAtDC`) to include user *Brian* in that list.

At this point, user *Brian* has a valid Kerberos ticket-granting ticket signed with the domain *krbtgt* account. Following step 5, *RODC01* will initiate steps 6 and 7 from before, except this time *RODC01* will attempt to cache user *Brian*'s password locally.

It is very important to realize, however, that the logon process for user *Brian* is not yet complete! Before Brian can use his workstation, he must obtain a Kerberos service ticket (*TGS*) for *PC01*. Continuing with our example, when *PC01* sends a TGS ticket to *RODC01* for user *Brian*, *RODC01* will be unable to decrypt that TGT because it was encrypted with the domain *krbtgt* account, whose password is *never* cached on an RODC. The following steps show the process for *Brian* to obtain a TGS ticket for *PC01*:

1. *PC01* sends a *KRB_TGS_REQ* to *RODC01* for user *Brian* to access *PC01*. The *KRB_TGS_REQ* includes *Brian*'s TGT from the previous example.
2. *RODC01* is unable to decrypt the TGT included in the TGS request since it is encrypted with the domain *krbtgt* account.
3. *RODC01* transmits the *KRB_TGS_REQ* to *RWDC01*, which replies with a *KRB_TGS_REP*.
4. *RODC01* receives a valid *KRB_TGS_REP*. Rather than forwarding the *KRB_TGS_REP* to *PC01*, *RODC01* transmits an error to *PC01*.

The specific Kerberos error that *RODC01* sends to *PC01* tells *PC01* that user *Brian*'s TGT has expired.

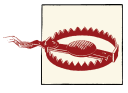
5. *PC01* prepares a new *KRB_AS_REQ* for user *Brian* and transmits it to *RODC01*.
6. Since *RODC01* now holds cached credentials for user *Brian*, *RODC01* is able to construct a new *KRB_AS_REP* locally (and thus a new TGT for Brian), encrypt it with *RODC01*'s local *krbtgt* account, and transmit it to *PC01*.
7. *PC01* sends a new *KRB_TGS_REQ* to *RODC01* for user *Brian*, this time including the new TGT for *Brian*.
8. *RODC01* is able to decrypt the TGT in the *KRB_TGS_REQ* and construct a *KRB_TGS_REP* that includes a service ticket permitting user *Brian* to use *PC01*.

You may be wondering what would happen if an RWDC received a TGT that was encrypted by an RODC. Each RODC's *krbtgt* account is replicated to all of the RWDCs in the domain, so RWDCs can decrypt Kerberos tickets encrypted by an RODC. The TGT includes unencrypted information in the header that allows an RWDC to determine which RODC's *krbtgt* account to use when decrypting the message.

After completing these steps, user *Brian* is able to use *PC01*. During subsequent logons by user *Brian* to *PC01*, the steps taken to obtain a service ticket (TGS) are much simpler:

1. *PC01* sends a *KRB_TGS_REQ* to *RODC01* for user *Brian* to access *PC01*. The *KRB_TGS_REQ* includes *Brian*'s TGT, which has been encrypted by *RODC01*.
2. *RODC01* decrypts the TGT in the *KRB_TGS_REQ* and constructs a *KRB_TGS_REP* that includes a service ticket permitting user *Brian* to use *PC01*.

After reviewing these examples, you may have begun to notice the importance of caching both the user and computer passwords on the RODC. In order for a user authentication to succeed, the RODC must possess the passwords for both the user and the computer. If either of these is not present, the RODC will need to communicate with a writable domain controller over the WAN. If the WAN is unavailable, the user logon will fail.



The writable domain controller that the RODC contacts must be running Windows Server 2008 or newer. If only Windows Server 2003 domain controllers are available, the logon request will fail.

Populating the password cache

Since the success of a logon processed by an RODC can be entirely dependent on whether or not the WAN (or more specifically, a Windows Server 2008 or newer writable domain controller) is available, you may want to prepopulate the RODC's password cache. You can prepopulate the password cache using the Prepopulate Passwords button shown in Figure 9-21. When you click this button, you will be able to search Active Directory for users and computers. After selecting the users and computers for which you wish to prepopulate the cache, you will be shown a confirmation dialog similar to Figure 9-25.

If any errors occur, you will be shown a dialog similar to Figure 9-26 advising you of those errors. Any principals whose passwords are successfully prepopulated will appear in the list shown in Figure 9-21.

Managing Cached Passwords with repadmin

The *repadmin* tool included with Windows Server 2008 and newer includes a number of switches that you can use to manage an RODC's password cache. You can use the */rodcpwdrepl* switch to specify one or more users or computers for which the RODC should immediately request replication of the password. The syntax for this command (all on one line) is: **`repadmin /rodcpwdrepl`** *`<RODC> <Hub DC> <user/computer distinguished name 1> <[user/computer distinguished name 2]...>`*

If, for example, you wanted to replicate the password for user *bdesmond* to an RODC called *K8DEVRODC01* from a RWDC called *K8DEVDC01*, you would run: **`repad`**

`min /rodcpwdrepl` *K8DEVRODC01 K8DEVDC01 cn=bdesmond,cn=users, dc=k8dev01,dc=brianlab,dc=local*

You can also use *repadmin* to copy all of the users and/or computers in the `msDS-AuthenticatedAtDC` to a group that is in the `msDS-RevealOnDemand` list. In order to run this command, you must specify the RODC and group using this syntax: *repadmin /prp move <RODC> <Group Name>*. The group name attribute expects the `sAMAccountName` value for a group, and if that group is not found, it will be created below the Users container in the domain. If you wanted to move all the users on the Auth-2 list for *K8DEVRODC01* to a group called *RODC01-Allow*, you could run *repadmin /prp move K8DEVRODC01 RODC01-Allow*. If you wanted to add only users or computers, you could append the */users_only* or */comps_only* switches.

Nearly all of the other PRP management tasks we cover in this chapter can be completed using *repadmin*. You can run *repadmin /prp /?* for a complete listing of PRP-related functions in *repadmin*.

Figure 9-25. Password cache prepopulation confirmation

RODCs and Write Requests

RODCs have a great deal of special logic encoded in them to handle write requests they receive from clients. Perhaps the most obvious and common example of a write operation is a user or computer password change request. Other common examples are dynamic DNS registrations and updates to logon timestamp information.

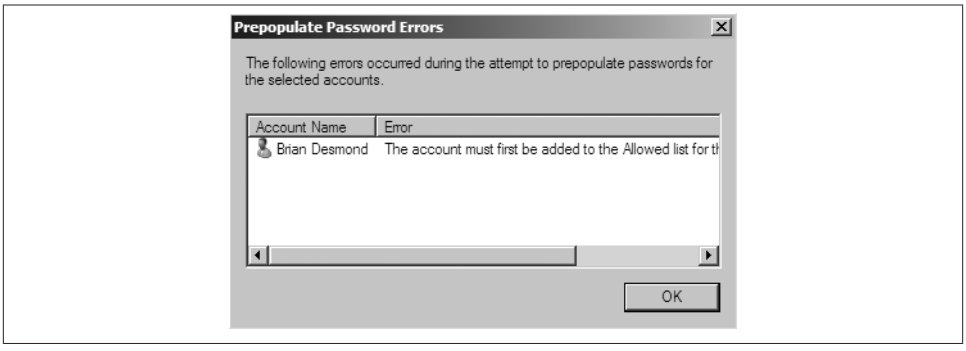


Figure 9-26. Password prepopulation errors

User password changes

The behavior of an RODC when a user changes his password varies depending on what mechanism the user is using for the password change (such as Kerberos or LDAP), and the operating system of the client. Let's take the most common example of a user pressing Ctrl-Alt-Delete on his workstation and selecting the "Change Password" option.

Clients running Windows Vista and newer (as well as Windows Server 2008 and newer) will attempt to make the password change via Kerberos. When an RODC receives a Kerberos password change request, it will be forwarded to a writable domain controller. The RODC will also immediately request that the RWDC replicate that user's password to the RODC. The RWDC will then follow the normal process discussed earlier for replicating a password to an RODC.

On a client running Windows XP or earlier (and server operating systems running Windows Server 2003 or earlier), the client will contact a writable domain controller directly and the RODC will have no knowledge of the password change. The next time the RODC replicates with a writable domain controller, it will erase its local copy of that user's password. The user authentication process discussed earlier ensures that the next time the user attempts to log on, the RODC will again cache that user's password if permissible.

If you have loaded the RODC hotfix package discussed in "Prerequisites" on page 231, the client will contact an RODC to change its password and the RODC will chain that request to an RWDC. The password will still not be requested for immediate replication, however.

If the user account password change request is made via LDAP (such as for the Outlook Web Access change password web page), the change will be made directly on a writable

domain controller. The RODC will have no knowledge of the password change, and the next time the RODC replicates with a writable domain controller, it will erase its local copy of that user's password. The user authentication process discussed earlier will ensure that the next time the user attempts to log on, the RODC will again cache that user's password if permissible.

When an RODC erases its local copy of a password, the old password is still physically stored in the Active Directory database. However, the replication metadata for the user is updated such that when queried, the database reports that the password field is null.

Computer account password changes

When a computer attempts to change its password, it will contact the RODC via its secure channel. The RODC will chain that password change request to a writable domain controller and request that the RWDC replicate the new password to the RODC. The RWDC will then follow the normal process discussed earlier for replicating a password to an RODC.

The lastLogonTimeStampAttribute

The `lastLogonTimeStampAttribute` sometimes defies the conventional wisdom we've been presenting throughout this chapter. If a user or computer's password is cached on the RODC, the RODC will locally write a new timestamp to the `lastLogonTimeStampAttribute` for that user or computer if the RODC determines that an update to the `lastLogonTimeStampAttribute` attribute is necessary. Simultaneously, the RODC will forward the `lastLogonTimeStampAttribute` update to a writable domain controller. Since the forwarded `lastLogonTimeStampAttribute` is slightly newer, the RODC's `lastLogonTimeStampAttribute` value for the user or computer will be overwritten with a slightly later value via normal replication.

The `lastLogonTimeStampAttribute` update will not be forwarded if the WAN is unavailable or the RODC is unable to contact a writable Windows Server 2008 domain controller

Last-logon statistics

If you have enabled tracking of last-logon statistics information as discussed in Chapter 16, the RODC will write updates to those attributes locally. However, unlike the `lastLogonTimeStampAttribute`, the RODC will *not* forward these updates to a writable domain controller.

Logon success/failure information

When an RODC processes a logon successfully, it will write new values for the `lastLogon`, `logonCount`, and `badPwdCount` attributes for the user or computer locally. As is the case with a writable domain controller, these attribute changes will never be replicated or chained to a writable domain controller. If an RODC processes a logon failure, it will update the `badPwdTime` and `badPwdCount` attributes. Likewise, these changes will never leave the RODC either by replication or by chaining.

NetLogon secure channel updates

Certain attributes of computer accounts (such as the computer's name, `dNSHostName`, and operating system information) that are stored in Active Directory are provided to domain controllers via a computer's NetLogon secure channel. When an RODC receives these updates, it will chain them to a writable domain controller. The RODC will initiate a request for immediate replication of the computer object, and if the computer's password is permitted to be cached, that request will succeed.

Replication connection objects

The Knowledge Consistency Checker (KCC) on an RODC will locally write the connection objects an RODC requires in order to replicate to the RODC's database. These changes will never replicate and are not necessary outside of the RODC.

DNS updates

RODCs leverage existing functionality in the DNS protocol in order to return a writable domain controller to the client. When a client is ready to make a dynamic DNS update, it will first contact the DNS server (in this case, the RODC) and request an authoritative name server for the zone it is trying to update by way of accessing the zone's SOA (Start of Authority) record.

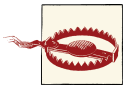
When an RODC generates the SOA record for a read-only DNS zone, it will attempt to locate a nearby Windows Server 2008 or newer writable domain controller that is a name server for the zone and that is also reachable. The RODC completes this process via a DC locator call. In the event that the RODC is unable to find a suitable Windows Server 2008 or newer domain controller, it will randomly select, from the list of available name servers, a name server that is not the RODC itself. The name server that the RODC ultimately selects during this process will be specified as the master server in the SOA record.

In order to keep the RODC's local version of the DNS zone up to date, the RODC will queue a request for replicating the client's record from the master server that the RODC selected for placement in the SOA record. Enqueueing this replication request is best-effort, which means that if it fails, the RODC will not try again to queue a replication

request for that DNS record. Instead, the RODC will receive the update during the next normal replication cycle.

If there is already an entry in the replication queue for the record in question, the RODC will not queue a second replication request.

By default, the RODC will process the replication queue every five minutes and attempt to replicate a maximum of 100 entries from the queue. For each entry in the queue, the RODC will verify that the entry is at least 30 seconds old, or else it will bypass it and attempt to replicate that entry during the next cycle. By default, when the queue depth reaches 300 entries, no more entries will be enqueued until the queue depth is below 300 entries.



The `DsPollingInterval` registry setting controls a number of activities in addition to the frequency of the DNS replication queue processing. These activities primarily include querying Active Directory for changes to AD-integrated DNS zones that are hosted on the DNS server. You should thus use caution when modifying this frequency.

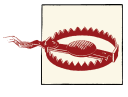
There are five registry settings that you can use to control the RODC DNS service's attempts to replicate dynamically updated DNS records. All of these registry settings are located under `HKLM\System\CurrentControlSet\Services\DNS\Parameters`. Table 9-3 details each of the registry settings you can configure.

All of the registry settings in Table 9-3 are `REG_DWORD` values.

Table 9-3. Read-only DNS registry settings

Registry value	Description	Unit	Default value	Min. value	Max. value
`DsPollingInterval`	Controls the frequency of the polling thread that the DNS service uses to query Active Directory for all AD-integrated DNS configuration settings.	Seconds	`180`	`30`	`3600`
`EnableRsoForRodc`	Controls whether the special DNS record-replication behavior is enabled or disabled.	Boolean	`TRUE`	`FALSE`	`TRUE`

Registry value	Description	Unit	Default value	Min. value	Max. value
`MaximumRodcRsoQueueLength`	Specifies the maximum number of entries that can be in the replication queue at any given time.	Entries	`300`	`1`	`1000000`
`MaximumRodcRsoAttemptsPerCycle`	Specifies the maximum number of records that the RODC will attempt to replicate in any given pass of the queue.	Entries	`100`	`1`	`1000000`
`DsRemoteReplicationDelay`	Indicates the minimum number of seconds an RODC will wait between enqueuing a replication request for a DNS record and processing the request.	Seconds	`30`	`5`	`3600`



The primary purpose of the `MaximumRodcRsoAttemptsPerCycle` setting is to limit the potential for a denial-of-service attack on a writable domain controller by way of flooding that domain controller with too many replication requests at once. Make sure to measure the impact on domain controller load if you increase this value.

In the event that you reach the maximum queue length (300 entries, by default), the RODC will log an event (event ID 4400, message "*The DNS server is experiencing high SOA query load. This may be caused by a large number of local client machines performing updates. Single object replication of DNS records corresponding to SOA queries is being throttled. This may delay replication of updates to this RODC DNS server, however scheduled replication will not be affected.*") in the DNS Server event log indicating that this situation has occurred, and no further replication requests will be enqueued until the queue is processed and its depth is decreased. If you are encountering this situation frequently, you may need to increase the maximum queue length by modifying the `MaximumRodcRsoQueueLength` registry setting.

The event log entry will be logged once per 24-hour period. If you restart the DNS server service, however, the event will be logged immediately following the next occurrence of the error condition.

The W32Time Service

As a rule, RODCs will synchronize time with writable Windows Server 2008 and newer domain controllers. An RODC will not synchronize its time with another RODC.

When an RODC receives a request for time synchronization from a client, the manner in which the request is processed varies greatly depending upon whether or not the RODC has the client's password cached. If the RODC has the client computer's password cached locally, the RODC is able to process and sign the time-synchronization request as a standard writable domain controller would. If, however, the RODC does not have the client's password cached, then a writable domain controller must be involved.

When the RODC receives a time-synchronization request it cannot process, there are a number of steps involved:

1. The RODC forwards the request to a Windows Server 2008 or newer writable domain controller.
2. The writable domain controller processes the request and signs it using the client computer's password, and then returns the response to the RODC.
3. The RODC in turn receives the response and forwards it to the client, who processes the response.

In order to handle the forwarding of requests and responses, the RODC maintains a local table of client time-service requests that it has forwarded. This table is known as the *chaining table*. The table contains three fields:

- The client's IP address (either IPv4 or IPv6)
- The timestamp from the NTP request
- The client's RID

When an RODC receives a reply from the writable domain controller, it searches the table for a matching request based on the RID and request timestamp. Once the RODC finds a match, it forwards the response back to the source IP.

In the event that the RODC is unable to find a match in the table, the RODC assumes that it originated the request. If the request is signed with the RODC's computer account password, then the RODC updates its local clock as necessary. Otherwise, the RODC discards the response.

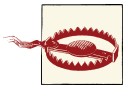
Any entry in the RODC's chaining table has a maximum lifetime of up to four seconds by default, and any given client (as defined by the client's IP address) can have a maximum of four entries in the chaining table at any given time. In total, the chaining table can contain a maximum of 128 entries by default. In the event that either of these maximums is exceeded, new requests are simply discarded until the chaining table has additional room for them.

Each time a new entry is added to the chaining table, the RODC will examine the chaining table and determine if any entries should be removed. RODCs will examine the chaining table for stale entries if more than 75% of the maximum number of entries are in use (96 entries by default). Any entry that is older than the maximum lifetime will be removed.

All of these defaults are configurable via a number of registry settings, shown in Table 9-4. These registry settings are located under `HKLM\System\CurrentControlSet\Services\W32Time\TimeProviders\NtpServer`.

Table 9-4. Chaining table configuration settings

Registry value	Description	Unit	Default value	Min. value	Max. value
`ChainDisable`	Controls whether the chaining of time requests by RODCs is enabled.	Boolean	`0`	`0` (chaining enabled)	`1` (chaining disabled)
`ChainEntryTimeout`	Controls the maximum lifetime of an entry in the chaining table.	Seconds	`4`	`4`	`16`
`ChainMaxEntries`	Controls the maximum number of entries in the chaining table.	Entries	`128`	`128`	`1024`
`ChainMaxHostEntries`	Specifies the number of entries any given host can have in the chaining table.	Entries	`4`	`4`	`16`



The purpose of controlling the size of the chaining table is to ensure that the chaining table does not exhaust memory resources on the RODC. If you increase `ChainMaxEntries`, take this warning into account.

If you disable the password chaining behavior on an RODC (by setting `ChainDisable` to `1`), the RODC will be able to communicate with down-level (e.g., Windows Server 2003) domain controllers in addition to writable Windows Server 2008 and newer domain controllers. If you make this configuration change, RODCs will no longer be able to process client time-synchronization requests.

Application Compatibility

As with any major change to your infrastructure, prior to large-scale deployment of RODCs in your environment, you will need to perform significant testing in order to understand application behavior issues as they relate to RODCs.

Applications that attempt to write to Active Directory using LDAP will receive an LDAP referral from an RODC to a writable domain controller. So long as the application is capable of *chasing* the referral, the write will succeed, and the next time the RODC

replicates, that write will be reflected locally. In the event that the WAN is down, the referral will fail. The application will need to be able to handle this situation.

You can disable an RODC's ability to issue write referrals by setting the `RODCMode` registry value to `1`. The `RODCMode` registry value is a `REG_DWORD` located at `HKLM\System\CurrentControlSet\Services\NTDS\Parameters`.

Another scenario where an application may fail is if it expects the results of a write to be immediately available on the local domain controller. One of the issues resolved by the RODC compatibility package is a bug in the Windows print-spooler service. The print spooler publishes printers to Active Directory via LDAP (resulting in a referral to a writable domain controller), and then immediately tries to read the published printers from the local domain controller. If the printer isn't present on the local domain controller (which is the case with a local RODC), the spooler service determines that publication of the printer has failed.

If an application is using legacy RPC calls to make changes to Active Directory, those RPC calls will fail if they are targeted to an RODC. The application must explicitly target a writable domain controller in order for the writes to succeed. The RODC will replicate the changes through normal replication.

Further, if you have added attributes to the filtered attribute set (FAS), applications will need to be aware that those attributes will not be available from Active Directory on an RODC. If an application requires access to an attribute in the FAS, the application will need to communicate with a writable domain controller. For more information on the filtered attribute set, see Chapter 5. Keep in mind that the FAS cannot be enforced in a multidomain forest until your forest is at the Windows Server 2008 or better functional level. This is because an RODC that is also a global catalog may replicate one of the Global Catalog naming contexts from a Windows Server 2003 domain controller. Windows Server 2003 domain controllers are unaware of the FAS, and as such they will include filtered attributes when replicating with an RODC. Filtering occurs at the replication source, not at the RODC, so the RODC will process inbound replication for filtered attributes and store them in its database.

In general, the most important thing you will need to do is test all of the applications in your environment that will potentially communicate with an RODC. You will likely find applications that require updates to their code or upgrades from the manufacturer. One application that you should be aware of is Microsoft Exchange. Exchange is unable to work with RODCs, and as such you should not collocate Exchange servers and RODCs in the same site.

Microsoft has compiled a list of applications that are known to work with RODCs at this link.

RODC Placement Considerations

When you place RODCs in your network, there are a few rules and considerations you should think about as you work on your design. The first rule to remember is that an RODC will never talk to another RODC.

If you're thinking of placing multiple RODCs in a given site, this is a very important rule to remember. Think of a situation where you have a branch office with two RODCs for the same domain. Let's call them *RODC01* and *RODC02*. Let's also assume you have a user *Brian* who works out of this site. On *RODC01*, Brian's password is cached; however, on *RODC02*, Brian's password is not cached. When the WAN becomes unavailable at this site and a writable domain controller is unable to be contacted, Brian attempts to log onto his workstation. Brian's workstation contacts *RODC02*, and because the WAN is unavailable and *RODC02* does not have Brian's password cached, logon fails. Brian's workstation will not try to contact a different domain controller and will instead simply tell Brian his password is invalid. Having multiple RODCs in a site can provide redundancy where necessary, but it can also provide a great deal of unpredictability. If you are planning to deploy multiple RODCs to a site, we recommend you prepopulate each RODC's password cache and maintain that cache with an automated process.

When you create a trust between a given set of domains, a special type of object called a `trustedDomain` is created in Active Directory to represent that trust. The `trustedDomain` object includes a password that is used for communicating with the remote domain in the trust. RODCs will never cache trust passwords, which means that any time a user who is in a site serviced by an RODC wishes to communicate with a resource in a trusted domain, the RODC must go to a writable domain controller to get the necessary Kerberos tickets. This is true even when the resources the user wishes to access are in the same site as the user. If the WAN is down, the user will be unable to access those resources since a writable domain controller for the user's domain cannot be contacted.

If you place RODCs in your perimeter network, you will need to pay special attention to applications that need to write to Active Directory as typically those writes will always fail. The assumption in this scenario is that you will close the firewall such that member servers in the perimeter network cannot communicate with writable domain controllers in the trusted network, since if you didn't do this, there would be no point in deploying RODCs in the perimeter network to begin with. Applications will fail to write to Active Directory since the RODC will issue a write referral to a writable domain controller that the client will be unable to contact. In this scenario we recommend you set the `RODC Mode` registry setting discussed earlier to `1`.

In order for dynamic DNS registrations to succeed for member servers in the perimeter network, you will need to allow perimeter network servers to contact writable domain controllers on TCP and UDP port 53. Domain join operations will fail for member servers in the perimeter network unless you precreate the computer account on a writable domain controller, ensure that the computer account is allowed to be cached by RODCs in the perimeter network, and then prepopulate the RODC password caches with that computer account's password.

In order for the prepopulation of the computer account's password to succeed, you will need to set it to a nonstandard value before you attempt to prepopulate the cache. To do this you can run *net user <ComputerName>$ <Password>*, substituting the computer's name and a password where appropriate. Be sure to include the trailing *$* in the computer name.

Administrator Role Separation

One often-sought-after feature for domain controllers is the ability to separate administration of the domain controller hardware platform and operating system from administration of the Active Directory service. On a writable domain controller, this is simply impossible since any administrators on the domain controller can make changes that could elevate their privileges across the forest and lead to compromise of the forest. RODCs introduce the ability to delegate administrative control of the operating system to a third party, and we highly recommend that you do this as a best practice, even if you are administering RODCs out of an Active Directory management team!

By delegating administrative rights to the RODC operating system to a group other than the Domain Admins group, you remove the need to ever log into an RODC as a domain administrator. Keep in mind the paradigm of treating an RODC as being compromised by default when you consider this recommendation. If you log into a compromised RODC with an account with domain admin rights, you could be giving away your domain admin password; or perhaps the OS has been modified to run a login script under your domain administrator login that makes compromising changes to the directory via an RWDC.

You can configure a multitude of different local roles on an RODC; however, we expect the most common local role you will configure is the Administrators role. The list of available roles includes:

- Administrators
- Users
- Guests

- Remote Desktop Users
- Network Configuration Operators
- Performance Monitor Users
- Performance Log Users
- Distributed COM Users
- IIS_IUSRS
- Cryptographic Operators
- Event Log Readers
- Certificate Service DCOM Access
- Incoming Forest Trust Builders
- Terminal Server License Servers
- Pre-Windows 2000 Compatible Access
- Windows Authorization Access Group
- Server Operators
- Replicator
- Account Operators
- Backup Operators
- Print Operators

In order to configure the local Administrators role on an RODC, you can use either *ntdsutil* or the Active Directory Users and Computers tool. ADUC limits you to configuring the Administrators role by specifying a user or group in the `managedBy` attribute of the RODC's computer account. Figure 9-27 shows this configuration. This change will become effective on the RODC when the updated `managedBy` attribute replicates to the RODC.

If you want to configure a role other than the Administrators role or you prefer to complete this configuration via the command prompt, you will need to use *ntdsutil*. To add a group called *COHOVINES\RODC01-Admins* to the local Administrators role on *RODC01*, you would run these commands after launching *ntdsutil*:

1. *local roles*
2. *connections*
3. *connect to server RODC01*
4. *quit*
5. *add COHOVINES\RODC01-Admins Administrators*

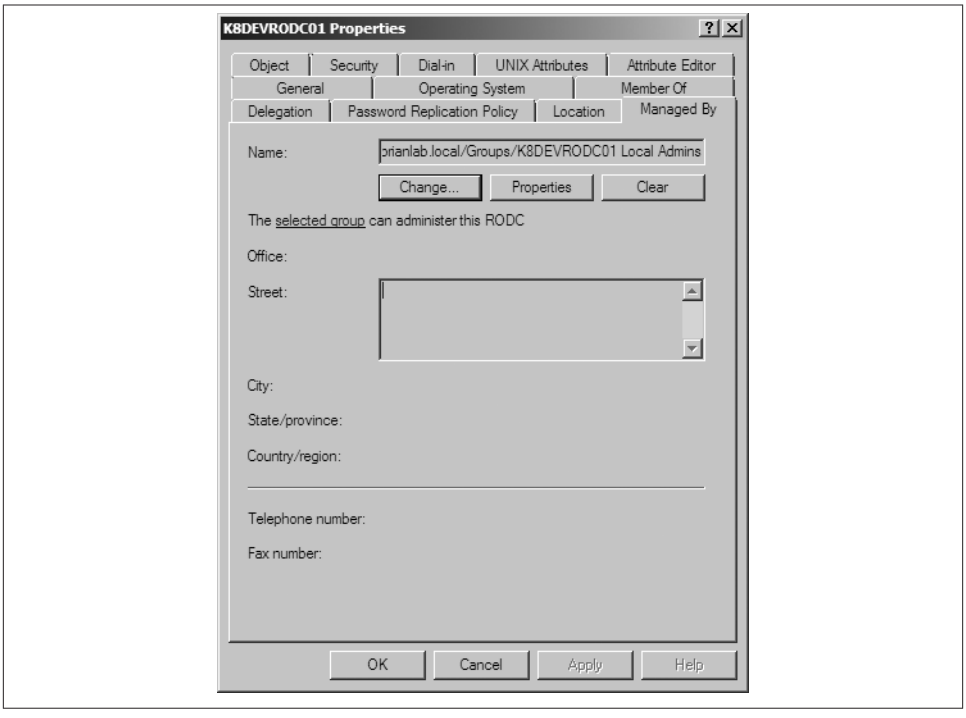


Figure 9-27. Configuring RODC local administrators in ADUC

The change will become effective immediately, and members of the RODC01-Admins group will be able to administer the RODC without concern that they will have administrative access to the domain or forest.

Users who are members of the local Administrators role can modify the members of this role and other local roles.

We highly recommend that you define a domain group that has local Administrators-level access to each RODC in your environment. This group should contain accounts that do not have any elevated privileges in your environment (i.e., normal user accounts). You should use these accounts for performing all administrative functions on an RODC. If you need to perform an administrative task that requires domain-level administrative permissions, perform that task remotely, as any attempt to log on interactively to the RODC either at the console or via Terminal Services could allow compromise.

Promoting an RODC

When you promote a server to be an RODC, you'll need to answer a few extra questions during the promotion process. Figure 9-28 shows the questions you'll be asked by Server Manager when you promote a new RODC.

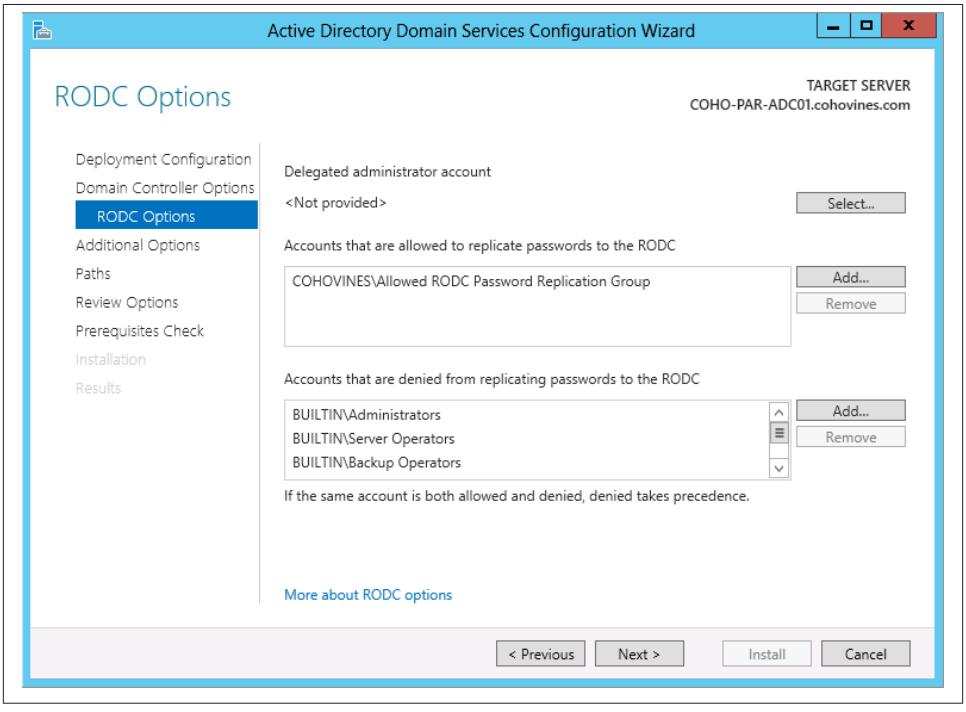


Figure 9-28. RODC promotion options

There are three questions in particular that you'll need to answer. First, you can optionally specify a group (or user) to delegate administrative rights over the RODC to, as discussed in the section "Administrator Role Separation" on page 253. You can always make this change later if you don't know at promotion time. Next, you can preconfigure the groups of users that are allowed to have their passwords cached on the RODC. The wizard preconfigures this list for you. Likewise, the wizard prepopulates the list of groups of users whose passwords cannot be cached. You can, of course, customize this as well.

If you'd like to promote an RODC from PowerShell with the parameters listed in Table 9-5, you can do so using this command:

```
Install-ADDSDomainController `
-AllowPasswordReplicationAccountName  `
@("COHOVINES\Allowed RODC Password Replication Group") `
-NoGlobalCatalog:$false `
-CreateDnsDelegation:$false `
-CriticalReplicationOnly:$false `
-DatabasePath "C:\Windows\NTDS" `
-DelegatedAdministratorAccountName "COHOVINES\COHO-TOK-ADC01 Delegated Admins" `
-DenyPasswordReplicationAccountName  `
@("BUILTIN\Administrators", "BUILTIN\Server Operators", `
"BUILTIN\Backup Operators", "BUILTIN\Account Operators", `
"COHOVINES\Denied RODC Password Replication Group") `
-DomainName "cohovines.com" `
-InstallDns:$true `
-LogPath "C:\Windows\NTDS" `
-NoRebootOnCompletion:$false `
-ReadOnlyReplica:$true `
-SiteName "Tokyo" `
-SysvolPath "C:\Windows\SYSVOL" `
-SafeModeAdministratorPassword  `
(ConvertTo-SecureString -AsPlainText -Force -String "password") `
-Credential (Get-Credential)
```

Table 9-5. RODC PowerShell promotion parameters

Parameter	Description
`AllowPasswordReplicationAccountName`	An array of groups or users that are allowed to have their passwords cached on the RODC.
`NoGlobalCatalog`	Whether this RODC should not be made a global catalog (watch out for the double negative here!).
`CreateDnsDelegation`	Whether a DNS delegation should be created.
`CriticalReplicationOnly`	Whether the promotion option should perform all replication or only critical replication. If you set `CriticalReplicationOnly` to `$true`, the server will reboot as soon as the minimum contents of the Active Directory database have been replicated locally. Replication will resume after reboot and the domain controller will advertise itself as available as soon as possible.
`DatabasePath`	Where to store the *ntds.dit* file.
`DelegatedAdministratorAccountName`	The user or group to delegate administrative access of the RODC to.
`DenyPasswordReplicationAccountName`	An array of groups or users that are not allowed to have their passwords cached on the RODC.
`DomainName`	The domain to promote the RODC into.
`InstallDns`	Whether the DNS server role should be installed locally.
`LogPath`	Where to store the database transaction log files.
`NoRebootOnCompletion`	Whether to restart the server when promotion is complete. If this is set to `$true`, the server will not reboot upon completion.
`SiteName`	The Active Directory site the domain controller should be promoted into.

Parameter	Description
SysvolPath	Where to store the Sysvol share.
SafeModeAdministratorPassword	The password for DS Restore Mode.
Credential	The credentials to use to access the target domain.

Prestaging RODC domain controller accounts

If you consider the compromised-by-default paradigm we discussed earlier in this section, you will probably conclude that using a domain admin account to promote an RODC is not the most secure choice. Fortunately, Active Directory has a solution for this, in the form of the *Add-ADDSReadOnlyDomainControllerAccount* PowerShell cmdlet. This cmdlet will do all of the work necessary as a domain admin ahead of time and allow you to delegate promotion of the RODC to any user or group in Active Directory. This feature has a twofold benefit of increasing security and allowing you to delegate promotion if you're going to be dependent on junior resources to complete the installation of RODCs.

The following command precreates an RODC account for *COHO-TOK-ADC01* in the Tokyo site of the *cohovines.com* domain. It also delegates promotion responsibilities to the COHOVINES\COHO-TOK-ADC01 Delegated Admins group.

```
Add-ADDSReadOnlyDomainControllerAccount `
 -AllowPasswordReplicationAccountName `
@("COHOVINES\Allowed RODC Password Replication Group") `
 -DomainControllerAccountName COHO-TOK-ADC01 `
 -DomainName cohovines.com `
 -DelegatedAdministratorAccountName "COHOVINES\COHO-TOK-ADC01 Delegated Admins" `
 -SiteName Tokyo `
 -DenyPasswordReplicationAccountName `
@("BUILTIN\Administrators", "BUILTIN\Server Operators", `
 "BUILTIN\Backup Operators", "BUILTIN\Account Operators", `
 "COHOVINES\Denied RODC Password Replication Group")
```

Once you precreate the RODC's account, you can run the cmdlet *Install-ADDSDomainController* (or the Server Manager wizard) with the -UseExistingAccount switch as a member of the Delegated Admins group to complete the promotion of the RODC.

If you're trying to prestage an RODC account on a version of Windows prior to Windows Server 2012, you should use Active Directory Users and Computers instead. To prestage the account, right-click the Domain Controllers OU in ADUC and click "Precreate Read-only Domain Controller account."

To use the prestaged account, run the following: *dcpromo /UseExistingAccount:Attach*.

Summary

In this chapter, we focused on how to deploy domain controllers, virtualization of domain controllers, and the read-only domain controller (RODC). Deploying domain controllers is a key step in building out an Active Directory installation and in the iterative lifecycle of the forest, as new versions of Windows become available and the organization upgrades Active Directory to the latest version.

Virtualization of domain controllers is a topic that has generated a great deal of concern and debate among Active Directory administrators for many years, and in Windows Server 2012, Microsoft addressed the technical challenges head-on. You can now safely roll back snapshots of DCs that are hosted on virtual machine generation ID-aware virtualization host platforms without concern for the impact on Active Directory. You can also clone virtual domain controllers to rapidly scale out your Active Directory infrastructure.

RODCs are a major feature in Active Directory, and they play a key part in the Active Directory deployments of many organizations. RODCs mitigate key security risks around Active Directory with regard to domain controllers that are in physically insecure locations, such as branch offices. Over the course of this chapter, we took a look at changes in behavior between RODCs and their writable counterparts.

CHAPTER 10

Authentication and Security Protocols

Active Directory is a core security component on the network, and one of the key protocols involved is *Kerberos*, a secure and flexible authentication protocol that we'll explore in detail in this chapter. Fortunately, Active Directory abstracts away most of the complexities of the protocol, so there are only a couple of configuration scenarios worth visiting. Those scenarios include service-specific configuration and Kerberos constrained delegation.

We'll wrap up with a look at a couple of security features that Active Directory brings to the table including authentication mechanism assurance and managed service accounts.

Kerberos

One of the fundamental underpinnings of any network that runs on Active Directory is the Kerberos security protocol. Kerberos provides the authentication mechanism that powers user logon, application access, and communication between domain controllers (among other things). Implementing Kerberos on its own is a challenging task that Microsoft has almost completely abstracted with Active Directory. Out of the box, there's virtually zero configuration required to start using Kerberos. In fact, if you never ran across an application that required special Kerberos-specific configuration, you would never even need to know that Kerberos was being used under the covers.

The key benefit of the Kerberos security protocol is the ability for a user to securely prove her identity and then achieve single sign-on to other services. In fact, with Kerberos, passwords never traverse the network in plain-text or encrypted formats. Instead, session-specific keys are generated for use over a short period of time through the use of various tickets. In this section, we'll look at how the protocol works for key scenarios.

User Logon

The first scenario that's likely to be of interest is how user logon occurs when a user sits down at her desk and presses Ctrl-Alt-Delete. We'll look specifically at the Kerberos layer of the logon process.

The key to the single sign-on benefit of Kerberos is the first step in the logon process: obtaining a *ticket granting ticket* (TGT). When a user logs onto a workstation, Windows caches the TGT and uses it to obtain service tickets without needing to prompt the user for credentials again when she wants to access a service on the network.

The first step in the authentication process is for the client to present a *preauthentication request* in an *AS_REQ* (authentication service request) packet. Figure 10-1 shows the contents of the *AS_REQ* message. The *AS_REQ* contains the necessary details to prove to the DC that the client is who it says it is. These details are:

Client Name
: This is the user's username.

Service Name
: This is the service principal name (SPN) of the *krbtgt* service on the domain controller.

Client Time
: To prove the user's identity, the current time on the client is encrypted with a *hash* of the user's password. The domain controller attempts to decrypt this field using the password hash stored in the Active Directory database. The timestamp is then checked to ensure that it is within tolerance (+/– five minutes, by default) to prevent replay attacks.

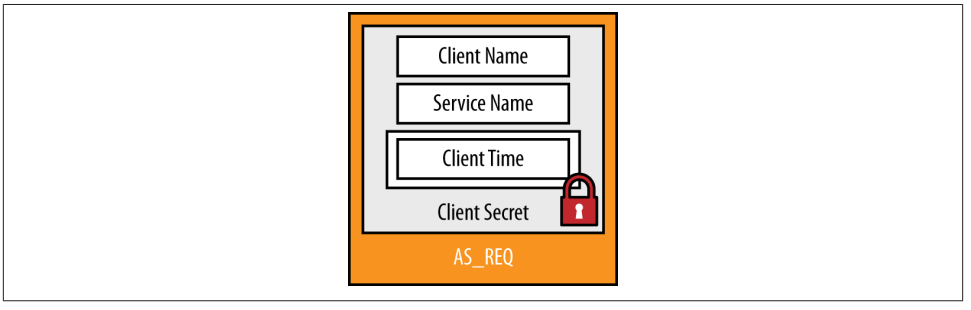


Figure 10-1. The authentication service request packet

If the user is logging on with a smart card instead of a username and password, a slightly different encryption process is followed. In lieu of encrypting the time with a hash of the user's password, the timestamp is encrypted with the user's private key (stored on the smart card). The DC uses the user's public key (stored in AD) to verify the validity of the signature on the request and thus authenticate the user.

Once the DC has validated the authentication request, it is ready to issue a TGT. The TGT is encapsulated in an *AS_REP* (authentication service response) packet, shown in Figure 10-2. The TGT contains the user's access token as well as a copy of the session key that is used for communication between the client and the DC going forward.

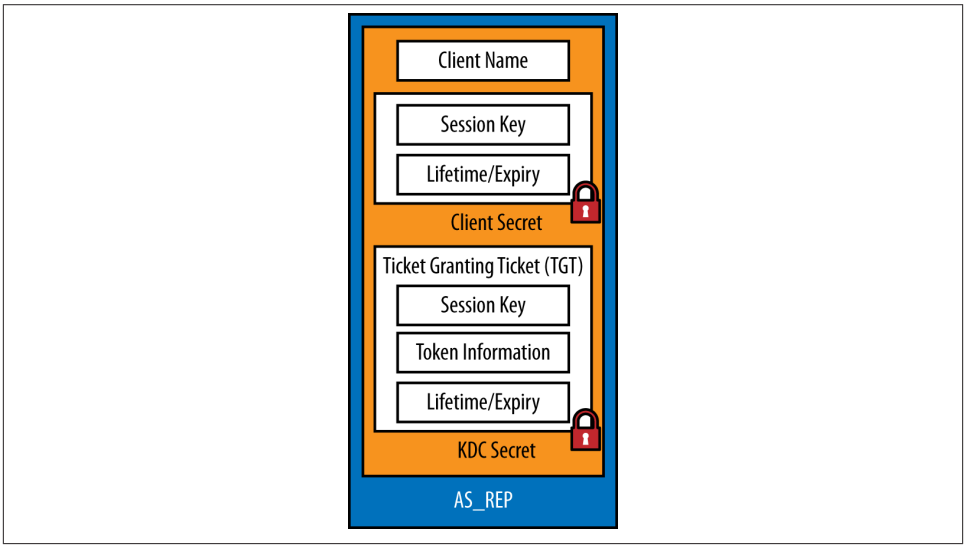


Figure 10-2. The authentication service response packet

The fields in the *AS_REP* message are:

Client Name
: This is the user's username.

Session Key
: This is a random cryptographic key that is used to secure future communications between DCs and the client. The session key is encrypted with a *hash* of the user's password.

Lifetime/Expiry
: TGTs have a defined lifetime (10 hours by default). After 10 hours, the TGT must be renewed or a new authentication request must be made. This field is also encrypted with a hash of the user's password.

Ticket Granting Ticket
: The TGT is the component that the client will cache in memory in order to request tickets to services. The entire TGT is encrypted with a hash of the KDC's secret, (specifically, a hash of the *krbtgt* account's password). This means that the TGT is opaque—clients can never see into the TGT.

  Inside the TGT, a copy of the session key is stored so that the DC can decrypt future communications with the client, as well as a copy of the ticket expiry information. Most importantly, the user's access token is stored in the TGT. The access token includes important information such as what groups a user is a member of, the user's NT rights, and Dynamic Access Control (DAC) claims.

Once the client receives the *AS_REP* message, the client decrypts the session key and caches it in memory alongside the TGT. At this point, the user's password is no longer necessary as all future Kerberos communications with the DC will be protected using the session key.

Service Access

Once a TGT is available, users can start accessing Kerberos-enabled services on the network transparently by simply requesting a service ticket for the service in question. The TGT obtained previously is the proof of authentication that the DC uses—thus eliminating the need for the user to provide her credentials again.

Kerberos-enabled (sometimes called *Kerberized*) services can be practically anything on the network. Typically these are things like file and print servers, websites, and databases, but the limit is only the ability of the application to accept authentication via Kerberos. Clients identify the service they want to access in the form of a *service principal name* (SPN).

Service principal names

Much like users are identified in the directory with a unique user principal name (UPN), services are identified with one or more unique service principal names. SPNs are the identifiers that clients use when they request a service ticket from a DC for a given service.

Fundamentally, an SPN is constructed in the form of a service identifier followed by the hostname—e.g., `service/host`. The service component is a predefined string that the client and server agree on—e.g., file shares are accessed using the `cifs` service name. The hostname can either be just the name of the server or service, or the FQDN of the

server or service. Often, both the short and long names are registered. For example, a web server called *WEB01.cohovines.com* would have the following SPNs registered in Active Directorycifs service name:

- `http/WEB01`
- `http/WEB01.cohovines.com`

SPNs can also have an optional port number if the server expects this. Microsoft SQL Server is an example of a product that adds the port to the SPN. A SQL server called *DB01.cohovines.com* would have the following SPNs registered in Active Directory:

- `MSSQLSvc/DB01:1433`
- `MSSQLSvc/DB01.cohovines.com:1433`

The most important thing to take away from this discussion of SPN construction is that the service and hostname components of the SPN are essentially arbitrary. The SPN that the client application constructs in order to request a service ticket simply must match the SPN that the server has registered in Active Directory.

SPNs are stored in Active Directory in the `servicePrincipalName` multivalued attribute. User, computer, and managed service accounts have the `servicePrincipalName` attribute. The SPN for a service must be registered on the account under which the service is running. If the application is running under a standard user or a managed service account, the SPN(s) should be registered on that object. If the application is running as *Local System* or *Network Service*, the SPN(s) should be registered on the computer object.

You can register SPNs using a tool such as ADSI Edit or the Attribute Editor in AD Users and Computers by simply adding them to the `servicePrincipalName` attribute of the object in question. You can also use the *setspn.exe* command-line tool.

The advantage of *setspn.exe* is that you can also perform a search for duplicates before registering the SPN. For example, to register the SQL Server SPNs discussed earlier on server *DB01,* where SQL Server is running under the *svc.DB01SQL* user account, you would run the following commands:

```
setspn -s MSSQLSvc/DB01:1433 svc.DB01SQL
setspn -s MSSQLSvc/DB01.cohovines.com:1433 svc.DB01SQL
```

You can also use the *setspn.exe* utility to unregister (remove) SPNs, search for existing SPN registrations, or check for duplicate SPNs across the forest. For a complete listing of parameters for *setspn.exe*, check out *http://bit.ly/WRjA0A*.

For a discussion of how to find duplicate SPNs and what happens when you have duplicates, check out *http://bit.ly/X4KlA9*.

Service tickets

Fundamentally, when the user decides to access a service for the first time, a service ticket request is constructed in a *TGS_REQ* (ticket granting service request) packet. The *TGS_REQ* contains the fields shown in Figure 10-3. These fields are:

Service Principal
: This is the SPN that uniquely identifies the service to which the user is trying to connect.

Client Name
: This is the user's username. This field is encrypted with the session key that was established during the authentication process.

Timestamp
: This field prevents replay attacks by ensuring that the request cannot be reused over a long period of time. By default, the request is only valid for +/– five minutes from the server's clock. This field is also encrypted using the session key.

Ticket Granting Ticket
: A copy of the TGT is included in the request. The DC uses this to build an access token for the service as well as to ensure that the user is in fact authenticated (and that the TGT has not expired).

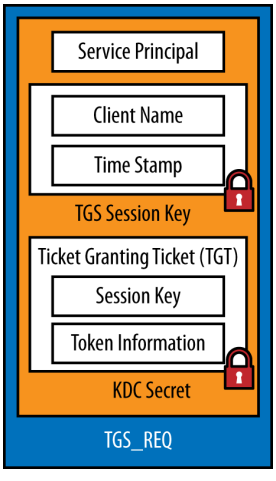


Figure 10-3. The ticket granting service request packet

When the DC receives the *TGS_REQ* message, it first attempts to match the Service Name field to a matching `servicePrincipalName` value in Active Directory. There are two steps to this process. First, a standard search is conducted to see if an explicit match can be found. If an explicit match isn't found, Active Directory checks for an implicit match.

Every account that has one or more SPNs with the `HOST` service name (e.g., `HOST/SRV01.cohovines.com`) receives several dozen additional SPNs automatically via the implicit matching process. The purpose of this process is to limit the amount of duplicate data. If this implicit matching process did not exist, the 52 entries in Table 10-1 would be duplicated on every single computer account in AD!

You can customize the list of implicitly mapped SPNs by editing the `sPNMappings` attribute of the `CN=Directory Service,CN=Windows NT,CN=Services` object in the Configuration NC. You can add additional mappings for the `HOST` service name, or add implicit mappings for another service name.

There are a few possible outcomes from the SPN lookup process:

- A match is found and a service ticket can be issued.
- Multiple matches are found (duplicate `servicePrincipalName` attribute values in the forest). In this case, an error is returned since Active Directory doesn't know which match is actually intended.
- No match is found. An error is returned.

The default list of SPNs that are implicitly mapped to `HOST` is provided in Table 10-1.

Table 10-1. Implicit HOST SPN mappings

alerter	fax	plugplay	schedule
appmgmt	http	policyagent	scm
browser	ias	protectedstorage	seclogon
cifs	iisadmin	rasman	snmp
cisvc	messenger	remoteaccess	spooler
clipsrv	msiserver	replicator	tapisrv
dcom	mcsvc	rpc	time
dhcp	netdde	rpclocator	trksvr
dmserver	netddedsm	rpcss	trkwks
dns	netlogon	rsvp	ups
dnscache	netman	samss	w3svc

```
eventlog     nmagent    scardsvr          wins
eventsystem  oakley     scesrv            www
```

Assuming everything checks out with the Service Name field in the *TGS_REQ*, the DC will validate the TGT (e.g., say that it isn't expired) and move forward with issuing the client a service ticket for the service in question. This ticket is issued in a *TGS_REP* (ticket granting service response) packet, shown in Figure 10-4.

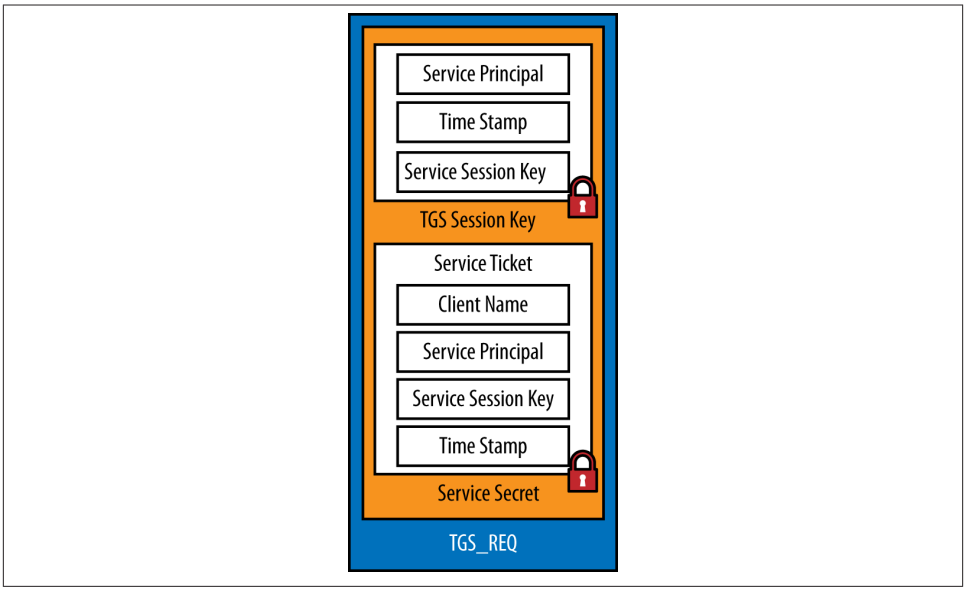


Figure 10-4. The ticket granting service response packet

The *TGS_REP* has the following fields:

Service Principal
: This is the SPN of the service for which the service ticket is valid. This field is encrypted with the session key that was established earlier during the authentication process.

Timestamp
: The timestamp of the message, used to prevent replay attacks. This field is also encrypted with the session key that was established earlier during the authentication process.

Service Session Key
: This is a second key that is cached by the client for encrypting communications with the specific service. This field is also encrypted with the session key that was established earlier during the authentication process.

Service Ticket

The service ticket is what the client will present to the service to request access and cache in memory for subsequent access to the service. Much like the TGT, the service ticket is opaque to the client—it is encrypted with a hash of the service's password.

The service ticket contains the name of the client, the service's SPN, a copy of the service session key that the client will use to encrypt communications with the service, and a timestamp.

Inside the service ticket is a copy of the access token that was calculated when the user's TGT was originally issued. The service can use this information (such as group membership) to authorize the user's access to the service.

Like the TGT, the service ticket contains an expiry date after which time the ticket will need to be renewed in order to continue.

Application Access

Once a client has a service ticket for a service, the client presents the ticket to the service in order to request access. The service ticket is presented to the service in an *AP_REQ* message. Unlike the messages we've discussed thus far, *AP_REQ* messages are presented in a different manner for every application. For example, for web applications the message is encoded in the HTTP headers, while LDAP expects it in the bind request.

Applications can also optionally support *mutual authentication* (whereby the service proves its identity back to the client) through the use of *AP_REP* messages. This is optional and not implemented in most cases.

Logon and Service Access Summary

Figure 10-5 summarizes the process we've discussed so far. To review, the following steps happen when a user logs onto his workstation and attempts to access a service:

1. An *AS_REQ* message is sent to begin the authentication process. This message proves the user's identity by encrypting a message with a *hash* of the user's password. Using a hash ensures that the actual password is never transmitted over the network.
2. The domain controller validates the request and produces a ticket granting ticket. The TGT is sent back in an *AS_REP* message. The client caches the TGT in memory and uses it for subsequent requests for service tickets.
3. The client sends a *TGS_REQ* message to the DC to request a service ticket for a specific service. Rather than providing credentials again, the client sends the TGT that it cached in memory after step 2.

4. The DC validates the *TGS_REQ* and constructs a service ticket for the service. The service ticket, encrypted with a hash of the services's secret, is sent back to the client in a *TGS_REP* message. The client caches this ticket in memory for subsequent use when authenticating directly to the service.
5. The client presents the service ticket to the service in an *AP_REQ* message. The service uses this to authenticate the user. The service might also use the user's access token (contained in the ticket) to perform authorization before allowing access.
6. Optionally, the service can respond with an *AP_REQ* message for mutual authentication of the service. This is not especially common and entirely optional.

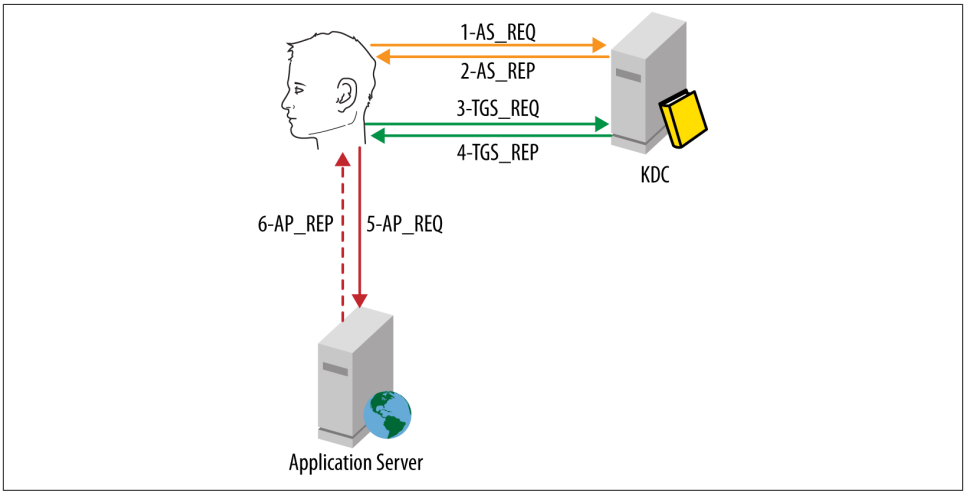


Figure 10-5. Kerberos message flow summary

As you follow this process, you can see how Kerberos enables single sign-on as well as a higher degree of security. Since the TGT is cached in memory on the client, the user's password is no longer necessary and can be discarded. Over the course of the TGT's lifetime, the client can present the TGT as many times as necessary to a domain controller to request a service ticket for a specific service on the network. Those service tickets are then cached in memory on the client and presented to the services any time the client needs to authenticate to them.

Delegation and Protocol Transition

One of the features of Kerberos that is often used by applications—especially web applications with a database tier—is called *Kerberos constrained delegation* (KCD). As we'll see in a minute, KCD enables an application to impersonate a user and make a request to another service as that user.

An extension of KCD is a feature called *protocol transition*, which is often used by single-sign-on gateways. In both cases, configuring these features can be a bit daunting.

Delegation

There are a couple of common models that applications use when they need to present (or modify) data from another system. The first model is known as the *trusted subsystem*, illustrated in Figure 10-6. With the trusted subsystem model, the user authenticates to the application (the web server, in this case), and then all of the authorization for the data is performed by the application. The application makes requests to the database using a shared service account.

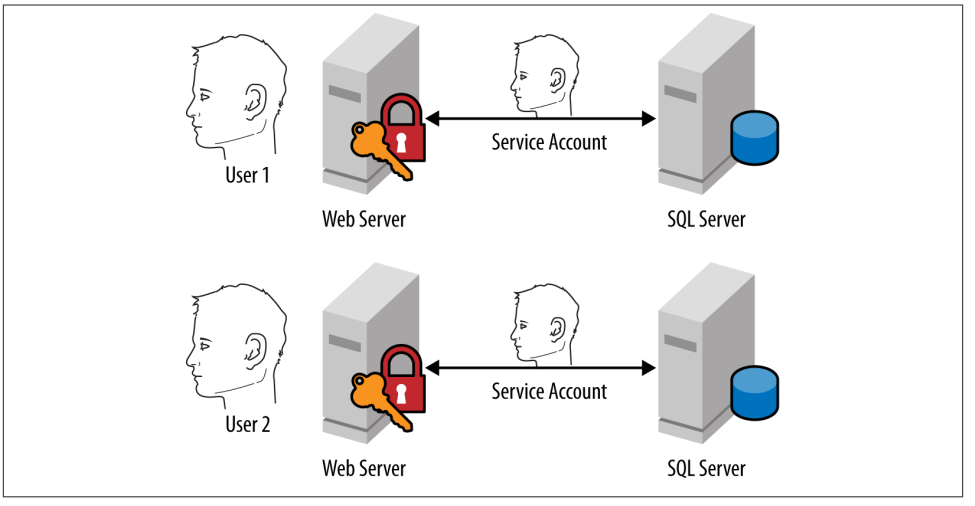


Figure 10-6. Trusted subsystem model

The downside to the trusted subsystem model is that the application is responsible for implementing an authorization model. The alternative is to depend on the backend system (a SQL server in this case) to perform the authorization and simply return the data that the user is allowed to access, as shown in Figure 10-7. In order to achieve this, the web server must impersonate each user when it connects to the backend SQL server.

To perform this impersonation, the web server needs the ability to request a service ticket for the SQL server for each user. In order to receive a service ticket, the web server completes the following steps (Figure 10-8):

1. The user requests a service ticket to the frontend service.
2. The user presents the service ticket to the frontend server as part of the request.
3. The frontend server submits the user's service ticket to the DC and requests a service ticket to the backend service for the user.

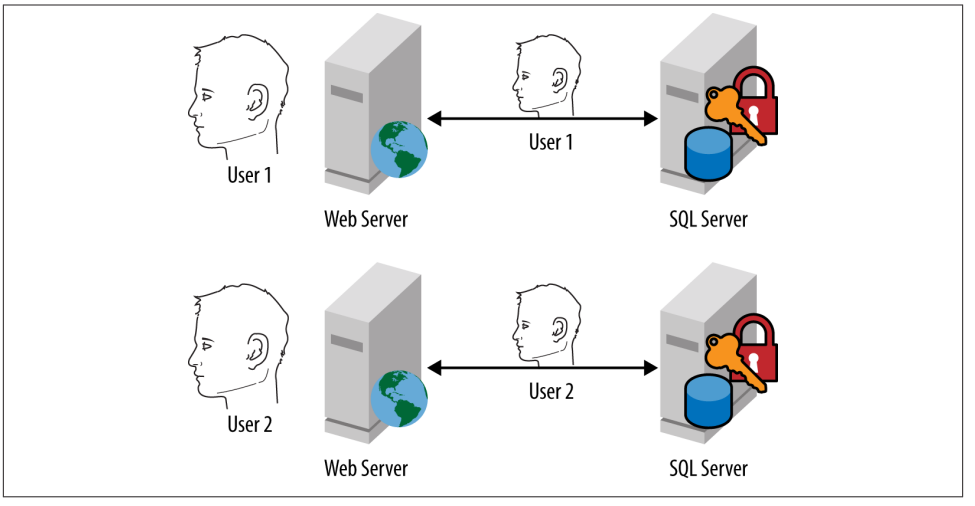


Figure 10-7. Delegated access to a backend service

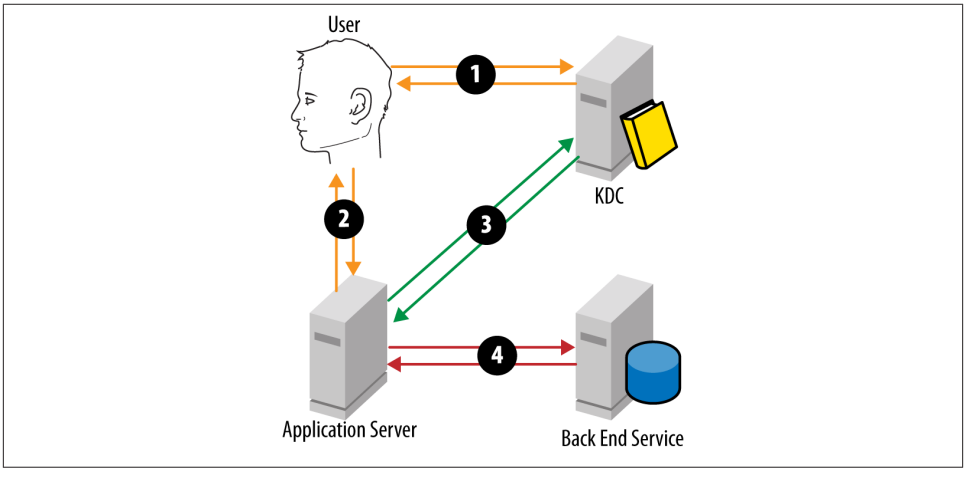


Figure 10-8. Kerberos message flow during delegation

4. The frontend server submits the service ticket to the backend service and interacts with the service in the context of the user.

This process is known as *delegation*. Since a third-party service is being given the right to impersonate a user and access another service, this is a sensitive operation. Consequently, services can't simply perform this operation without authorization.

Authorization is granted in Active Directory in the form of an allowance for a service to delegate to other specific services. This is known as *constrained delegation*. The easiest way to configure delegation is via the Delegation tab in the AD Users and Computers (ADUC) MMC snap-in.

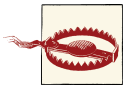
Delegation is configured on the frontend service's account as a permission to delegate to one or more SPNs. Remember, this is the service account (or managed service account) if the service is running as that account, and the computer account if the service is running as *Local System* or *Network Service*.

Consider a scenario where a web application running as *Network Service* on the *COHO-CHI-WEB01* server needs to delegate to a SQL server with an SPN of `MSSQLSvc/DB01.cohovines.com:1433` running under the *svc.DB01Service* user account. To configure this, find the *COHO-CHI-WEB01* computer account in ADUC:

1. Select the Delegation tab shown in Figure 10-9.

If you are configuring delegation on a user account, the Delegation tab is only visible if one or more SPNs are registered on that user or the user account is already configured to allow delegation to a service. You can add a temporary bogus SPN to the account, or you can add the SPNs that can be delegated directly to the `msDS-AllowedToDelegateTo` attribute.

2. Select "Trust this computer for delegation to specified services."



The "Trust this computer for delegation to any service (Kerberos only)" option shown in Figure 10-9 is called *unconstrained delegation*. This was the only option supported in Windows 2000 and is highly dangerous. If you enable this option, the frontend service can impersonate an authenticated user to any service on the network!

You should not use unconstrained delegation without thorough risk review.

3. Click Add and then click "Users or Computers".
4. Find *svc.DB01Service*. Click OK.
5. Select the SPN you want to delegate to, as shown in Figure 10-10.
6. Click OK.

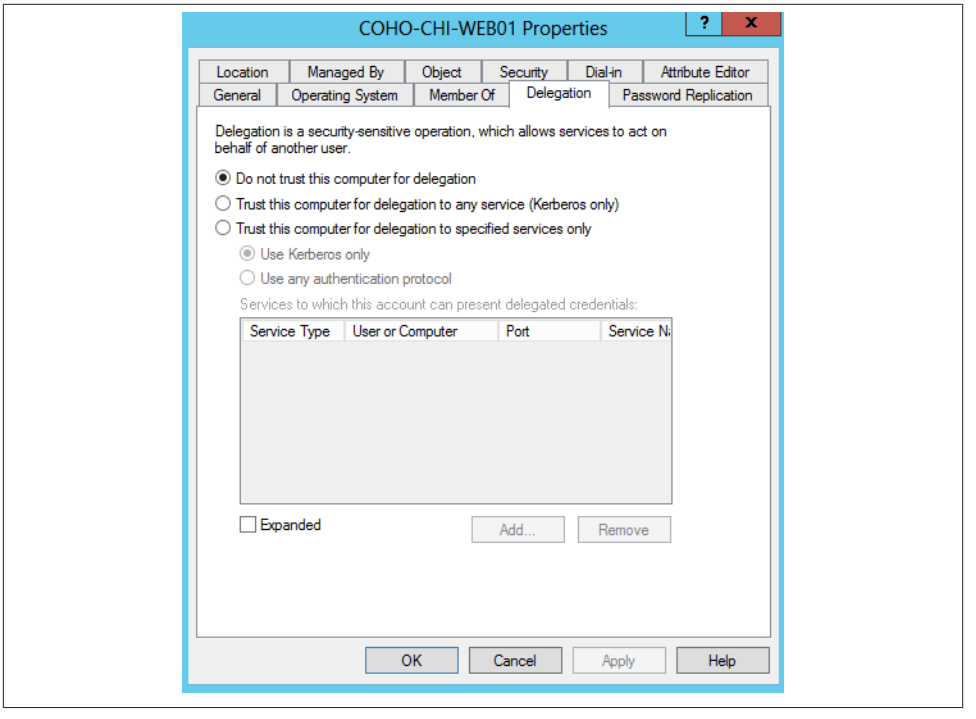


Figure 10-9. Service delegation configuration

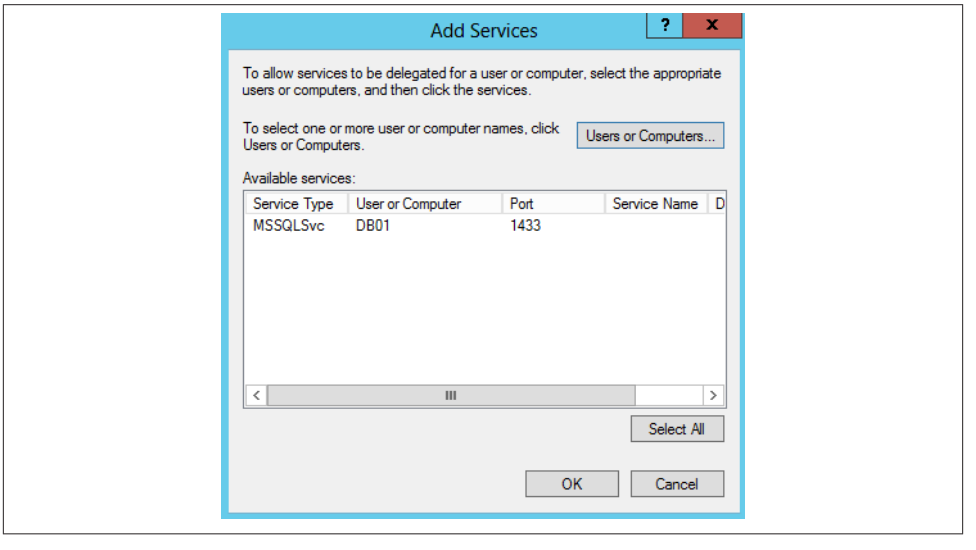


Figure 10-10. Adding services for delegation

This is all that's necessary to allow the *COHO-CHI-WEB01* server to delegate to the `MSSQLSvc/DB01.cohovines.com:1433` SPN. If you are running Windows Server 2012 and you meet the following requirements, you can also configure delegation in a different way using Windows PowerShell:

- The frontend service is running on a Windows Server 2012 or Windows 8 machine.
- The backend service is running on a Windows Server 2003 or better machine.
- At least one Windows Server 2012 domain controller exists in the frontend service's domain.
- At least one Windows Server 2012 domain controller exists in the backend service's domain.

While we discussed previously that you would configure the list of principals that the frontend service is allowed to delegate to, with the new model, the backend service is configured with a list of principals that are allowed to delegate to it. You can now also delegate permissions in Active Directory to configure constrained delegation.

To configure the scenario discussed earlier, you would run the following PowerShell command:

```
Set-ADUser svc.DB01Service -PrincipalsAllowedToDelegateToAccount COHO-CHI-WEB01$
```

This model also enables KCD across domain boundaries. Previously, KCD could only be performed when both the frontend and backend service identities were located in the same domain.

Protocol Transition

Protocol transition is a highly sensitive variant of KCD that enables delegation to be performed without presenting the user's service ticket to the domain controller to prove that the user authenticated to the frontend service. This feature is commonly used by web-based single-sign-on applications that have a friendly forms-based authentication. Rather than presenting a service ticket to the application, users authenticate to the application directly, perhaps by entering their Active Directory credentials or another set of credentials that the application recognizes.

Next, the application makes a request to Active Directory for a service ticket in the user's name for a specific service. Since this operation requires the frontend service to be trusted to request service tickets for users without proving that the user has authenticated, additional access must be granted. This access is granted by selecting the "Any Protocol" option in Figure 10-9 when KCD is configured.

Authentication Mechanism Assurance

Authentication mechanism assurance (AMA) is a feature that was introduced in Windows Server 2008 R2 to provide conditional access to resources based on the smart card certificate that a user uses to access the network. Typically, the feature is used when different types of certificates are issued to users based on the level of assurance that has been reached about the user's identity. For example, one level of assurance might simply be tied to verifying a government-issued photo ID for the user when she is hired, while another level of assurance might involve two forms of ID and a background check.

When smart card certificates are issued, they include object identifiers (OIDs) that identify various properties of the certificate. One of these properties might be the level of assurance that correlates to the user to whom the certificate was issued.

With AMA, Active Directory can dynamically inject the SID of a group into a user's access token at logon based on the presence of an OID in the user's smart card certificate. This can be used to dynamically secure resources such that they can only be accessed when a user is accessing the network with a certificate with specific properties. You can then check for the presence of this group in access control lists, applications, and so forth.

Since this feature is not often implemented, we refer you to Microsoft's Step-by-Step guide for implementing AMA if you want to use the feature. The guide is available at *http://bit.ly/YDlBev*.

Managed Service Accounts

Service accounts represent a gap in the security plan for many organizations. Normal user accounts typically have a well-defined password policy as well as a deprovisioning process associated with them. Service accounts, on the other hand, often have neither of these key security strategies applied to them. It is also not atypical for IT organizations to take shortcuts such as sharing service accounts between applications or using a common password across multiple service accounts.

All of these issues lead to a security gap in the area of service accounts. When IT personnel leave the organization, knowledge of service accounts and their passwords leaves with those employees. Since those passwords are often set to never expire and also rarely (if ever) change, and service accounts often have elevated access, there is an obvious security hole related to ex-employees' continued potential ability to access these accounts. Service accounts also provide an attractive target for an attacker, given the accounts' roles in the organization's IT infrastructure.

Managed service accounts (MSAs) are a feature Microsoft first introduced in Windows Server 2008 R2 to help mitigate this risk. Windows Server 2012 introduces a much improved version of MSAs, called *group managed service accounts* (gMSAs). In both

cases, the MSA is a special type of security principal that services, IIS application pools, and in some cases scheduled tasks can run as. Rather than having a conventional password, MSAs (and gMSAs) establish a secret with the directory and then rotate that password every 90 days (by default). This mitigates the risk involved with service accounts.

Unfortunately, the first iteration of MSAs in Windows Server 2008 R2 had a key limitation: the service could only run on a single server at a given time, since there was no mechanism to communicate passwords between servers. This limited the utility of MSAs, in clustering and load balancing scenarios in particular. Fortunately, the gMSAs introduced in Windows Server 2012 solve this problem.

Preparing for Group Managed Service Accounts

Before you can begin using gMSAs, you need to prepare Active Directory. You'll need to have at least one Windows Server 2012 domain controller (DC) in the domain in which you want to create the gMSA. Windows Server 2012 DCs run the new Group Key Distribution Service (GKDS). The KDS is responsible for calculating gMSA passwords and providing those passwords to authorized clients.

gMSAs can authenticate to any DC. Windows Server 2012 DCs are only required for password management operations.

Before you can begin using gMSAs in a domain, you must create a KDS root key for that domain. You can do this with the *Add-KdsRootKey* PowerShell cmdlet. By default, a KDS root key generated by simply running *Add-KdsRootKey* is not effective for a period of 10 days. This is designed to allow the key to propagate to all Windows Server 2012 DCs before it becomes effective.

If you want to make a key available immediately, run *Add-KdsRootKey -EffectiveImmediately*. You can also modify the effective date at creation time. If, for example, you know that your domain will converge within eight hours, you can create a key that is effective in eight hours by running the following commands:

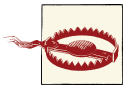
```
$effectiveDate = (Get-Date).AddHours(8)
Add-KdsRootKey -EffectiveDate $effectiveDate
```

Once you have generated a root key and it has become effective, you can start taking advantage of gMSAs.

Using Group Managed Service Accounts

In order to use a gMSA, you must be using a Windows 8 or Windows Server 2012 client. gMSAs are supported for conventional Windows services, IIS application pools, and

scheduled tasks. Very few applications support natively specifying a gMSA during setup, so chances are you'll need to install the application using a conventional service account and then change the application's identity later using the Services MMC snap-in.



While this trick should work for most applications, some applications may take a dependency on the SID or username of the service account specified during setup. In this case, changing the service account using the Services snap-in would break the application.

As an example, we'll configure an IIS application pool to use a new gMSA. The first step is to provision the gMSA. There is currently no GUI option available, so this task must be completed via Windows PowerShell. To create the gMSA, run the following command:

```
New-ADServiceAccount `
-SamAccountName svc.cohowebsite `
-Name svc.cohowebsite `
-DisplayName "Coho Vines Website" `
-DNSHostName "www.cohovines.com" `
-PrincipalsAllowedToRetrieveManagedPassword `
@("COHO-CHI-WEB01$", "COHO-CHI-WEB02$")
```

In this command, we specify the following parameters:

SamAccountName
: The username to assign to the service account. Windows automatically appends a dollar sign (*$*) to the end of the account.

You may be wondering why Windows appends a *$* to the *SamAccountName* value you specify. A piece of trivia that may help further your understanding of MSAs (and gMSAs) is that they are a derivative of the `computer` object class in the schema. This means that they inherit all of the attributes of the `computer` class in addition to the attributes specific to the MSA object classes.

Name
: This corresponds to the `name` attribute of the object in Active Directory. We generally use the same value as the *SamAccountName* for this field.

DisplayName
: A friendly name to show in management tools.

DNSHostName
: A DNS FQDN corresponding to the service.

If this parameter is not applicable to your scenario, you can enter a bogus value.

PrincipalsAllowedToRetrievePassword
: A list of objects in AD that can retrieve the password for the account. This is generally a group containing the computers the service is running on, or a list of computer objects. In the example, we have specified the computers the service will run on.

If you use a group to control access to the gMSA, you will need to reboot the group members before they can start using the gMSA.

In addition to the parameters listed here, there are a number of other useful parameters of the *New-ADServiceAccount* cmdlet. You can review a complete listing at *http://bit.ly/ZQjFll* or by running `Get-Help New-ADServiceAccount -Detailed`.

Once you have created the MSA, you must install it on the server(s) that will be using the MSA. This is a straightforward process achieved with the *Install-ADServiceAccount* cmdlet. The *Install-ADServiceAccount* cmdlet is part of the Active Directory PowerShell module. If you have not already installed this module on the server that you'll be installing the MSA on, run `Install-WindowsFeature RSAT-AD-PowerShell` to do so.

To install the gMSA we created earlier, run `Install-ADServiceAccount svc.cohowebsite`. Next, configure the IIS application pool. In order to signify that this is an MSA, you need to append a dollar sign to the username and leave the password fields blank, as shown in Figure 10-11.

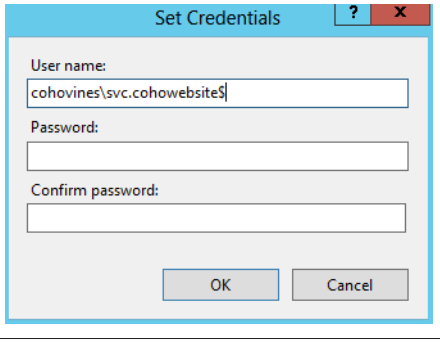


Figure 10-11. IIS application pool credential configuration

For services, you should take the same approach of specifying the username with a *$* appended and leaving the password fields blank. Scheduled tasks are a bit more involved, since the Task Scheduler GUI doesn't support MSAs. You'll need to create the scheduled task using the command line instead. The following sample creates a task that runs a fictitious batch script called *C:\Scripts\ScheduledBatch.bat* every morning at 3 A.M. using the *svc.BatchJob* gMSA (each command should be on a single line):

```
$taskAction = New-ScheduledTaskAction -Execute "C:\Scripts\ScheduledBatch.bat"
$taskSchedule = New-ScheduledTaskTrigger -Daily -At 3AM
$taskIdentity = New-ScheduledTaskPrincipal -UserId COHOVINES\svc.BatchJob$ `
-LogonType Password

Register-ScheduledTask -TaskName "Scheduled Batch" -Action $taskAction `
-Trigger $taskSchedule -Principal $taskIdentity
```

For more information about each of the cmdlets used in this example, refer to these links:

- New-ScheduledTaskAction
- New-ScheduledTaskTrigger
- New-ScheduledTaskPrincipal
- Register-ScheduledTask

You will also need to grant your MSA the "Log on as a batch job" Windows right before the scheduled task will start working. You can do this either via Group Policy or the server's local security policy. To make this change on the local server, launch the Local Security Policy Editor (Start→Run→**`secpol.msc`**) and browse to *Local Policies\User Rights Assignment*, as shown in Figure 10-12. Add the MSA to the "Log on as a batch job" right.

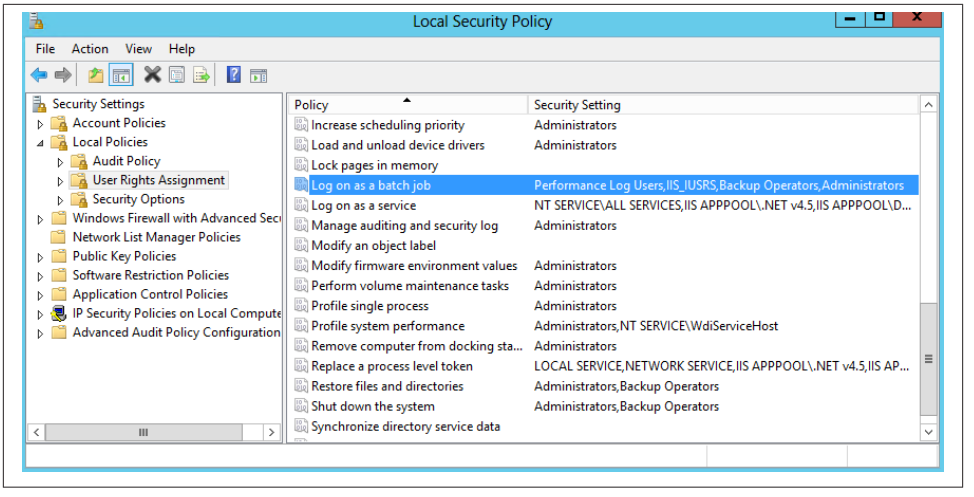


Figure 10-12. Local Security Policy Editor

Summary

In this chapter, we explored a key feature of the overall Active Directory infrastructure—Kerberos. The Kerberos security protocol is used for authentication to the network and to services on the network. We also looked at a common Kerberos configuration requirement called Kerberos constrained delegation. Finally, we looked at Active Directory security features such as authentication mechanism assurance and managed service accounts.

CHAPTER 11
Group Policy Primer

Group Policy is a large topic that deserves a book in itself (and there are several of those) to be properly covered. We will discuss Group Policy as it applies specifically to the design and administration of an Active Directory installation in this book, but not as it applies to the actual settings and operations on a workstation.

The goal of policy-based administration is for an administrator to define the environment for users and computers once by defining policies, and then to rely on the system to enforce those policies. This chapter is an introduction to Group Policy and how to manage it. Chapter 15 covers how to begin designing Group Policy and the OU structures in support of Group Policy.

The scope and functionality of Active Directory group policies encompass a number of key points:

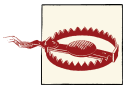
- They can be targeted to individual computers and users, sites, domains, and organizational units.
- They can apply to users, computers, or groups of either.
- They can set values and automatically unset them in specified situations.
- They can run scripts at user logon and logoff and computer startup and shutdown.
- They can do far more than just a desktop lockdown.

With Group Policy, administrators can control the behavior of workstations and servers as well as managing the end user experience across the organization. There are literally tens of thousands of settings that you can apply to control everything from screensaver timeouts to desktop backgrounds to workstation power management, and practically everything in between. We'll take a look at the mechanics of applying these settings and how things work under the hood with regard to Group Policy in this chapter, but we won't look at the actual settings you can apply.

Capabilities of Group Policy Objects

The Group Policy Management Console (GPMC) is the best tool for managing group policy objects (GPOs). In addition to the GPMC, the Group Policy Management Editor (GPME) MMC snap-in is used for editing the actual settings in GPOs. The GPMC provides a single interface to manage all aspects of GPOs, including editing (through the GPME), viewing the resultant set of policies (RSoP), and linking to domains, sites, and OUs. If you're still using Windows XP or Windows Server 2003 to manage your group policies, you'll need to download and install the GPMC on your machine. The GPMC installation is available from *http://bit.ly/YFv199*. Newer versions of Windows include the GPMC as an optional feature or in the Remote Server Administration Tools (RSAT) download. If you are using the GPMC on Windows 7 or Windows Server 2008 R2, the hotfix available at *http://support.microsoft.com/kb/2466373* corrects an annoying problem that affects it in some cases.

There are four different policy areas that a GPO supports. These areas are registry settings, security settings, group policy preferences, and software installation. Most registry 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. In some cases the setting needs no other parameters, while in other cases a host of information must be entered to configure the setting; it all depends on what the setting itself does.



Enabling and disabling most settings is fairly straightforward. However, due to Microsoft's choices for the names of certain settings for GPOs, you actually can have the choice of enabling or disabling options with names such as "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 widely deploy the GPO.

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 GPME under the headings of Policies and Preferences. There are two sets of these headings, one under Computer Configuration and one under User Configuration. The items under the two headings differ, as the settings that apply to users and to computers are not the same.

Group Policy Storage

Group policies are stored as a number of components that collectively enable the Group Policy functionality both at the client side and for the administrator. This data is

replicated using a number of methods, depending on the component and platform. Throughout this section, we'll discuss the key components of group policies as well as the replication engines available and how to work with them.

ADM or ADMX files

Some of the settings under Administrative Templates might seem to make more sense if they were grouped in one of the other sections. However, the Administrative Templates section holds the settings that are entirely generated from the administrative template (ADM) files in the system volume (SYSVOL), 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 GPME.

Typically, a large portion of the disk space consumed by a domain's SYSVOL is accounted for by the ADM files. ADM files don't play a part in the application of group policies, but rather are there solely to populate the Group Policy Object editor and GPMC reports. The Group Policy toolset included with Windows Server 2008 and newer supports a new type of ADM file that has an ADMX extension. ADMX files serve the same purpose as ADM files; however, they are in a new XML format that cannot be interpreted by older versions of the Group Policy tools. Each ADMX file that Microsoft provides is accompanied by ADML files for each language Windows supports. Localized strings for different languages can be stored in separate ADML files, which are referred to by the accompanying ADMX file. So, English-speaking users can see Administrative Template policy descriptions in English, whereas a French-speaking user, looking at the same GPO, will see those descriptions in French.

Microsoft has a free tool called the ADMX Migrator available for converting custom ADM files into the new ADMX format, as well as for graphically creating new ADMX files. You can get this tool at *http://bit.ly/13tdQiR*.

Windows Server 2008 and newer Group Policy tools also support the concept of a central store for ADMX files. The central store eliminates the duplicate storage of ADM files within each group policy and instead stores them once per domain. This can greatly reduce the amount of disk space the domain's SYSVOL share requires and also decrease replication requirements. You will need to create the central store manually in order to utilize this feature. Fortunately, creating the central store is quite straightforward:

1. Create a folder called *PolicyDefinitions* in the SYSVOL policies subfolder. If, for example, your domain was called *mycorp.com*, you would create *PolicyDefinitions* under *mycorp.com**sysvol**mycorp.com**policies*\.

2. Copy the contents of the *%SystemRoot%\PolicyDefinitions* folder into the *PolicyDefinitions* folder you created in step 1. You should obtain the contents of the *PolicyDefinitions* folder from a machine with the latest version of Windows and the latest accompanying service pack.

You should only use Group Policy tools from the corresponding version of Windows and Windows Server. If, for example, you are using the Windows Server 2012 Group Policy tools, do not attempt to edit policies using the Windows Server 2008 Group Policy tools. There are incompatibilities in ADMX file formats between the versions.

3. Restart any open Group Policy tools. The Group Policy Editor and the GPMC will automatically begin utilizing the central store.

Keep in mind that Windows Vista SP1 and Windows Server 2003 or earlier Group Policy toolsets cannot leverage ADMX files or the central store, so in order to use the central store you must migrate to editing group policies with the newer tools. To take advantage of the central store, you only need to upgrade your Group Policy management tools. It does not matter what version of Windows your domain controllers are running, but if they are running Windows Server 2003, you should no longer use the Group Policy tools that are included with Windows XP and Windows Server 2003.

Once you have set up the central store, ADM files will not be cleaned up automatically. You will need to do this by hand. The blog post at *http://bit.ly/XDRTWl* has a sample script to help you clean up old ADM files.

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. Doing 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 it. 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 DC's system volume (SYSVOL), 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 apply the group policy. In other words, one object can be linked to multiple locations in the tree, which explains how one GPO can be applied to as many organizational units, sites, or domains as required.

Let's consider the `groupPolicyContainer` class objects themselves. Take a look at Figure 11-1; we are using ADSI Edit to show the view of the Policies container and its children. Here you can see a number of `groupPolicyContainer` objects shown with a GUID as the `cn` field. The `displayName` attribute of these objects holds the name that administrators of Active Directory would see when using one of the normal tools to view these objects. Each GPO also has a `gPCFileSysPath` attribute that holds the full path to the corresponding directory in the system volume.

If you were to look under the Policies container on a default installation, you would find only two children. These children would correspond to the Default Domain Policy and the Default Domain Controllers Policy, the only GPOs created automatically by the system on installation. These GPOs have fixed names across all domains:

- *{31B2F340-016D-11D2-945F-00C04FB984F9}* is always the Default Domain Policy.
- *{6AC1786C-016F-11D2-945F-00C04fB984F9}* is always the Default Domain Controllers Policy.

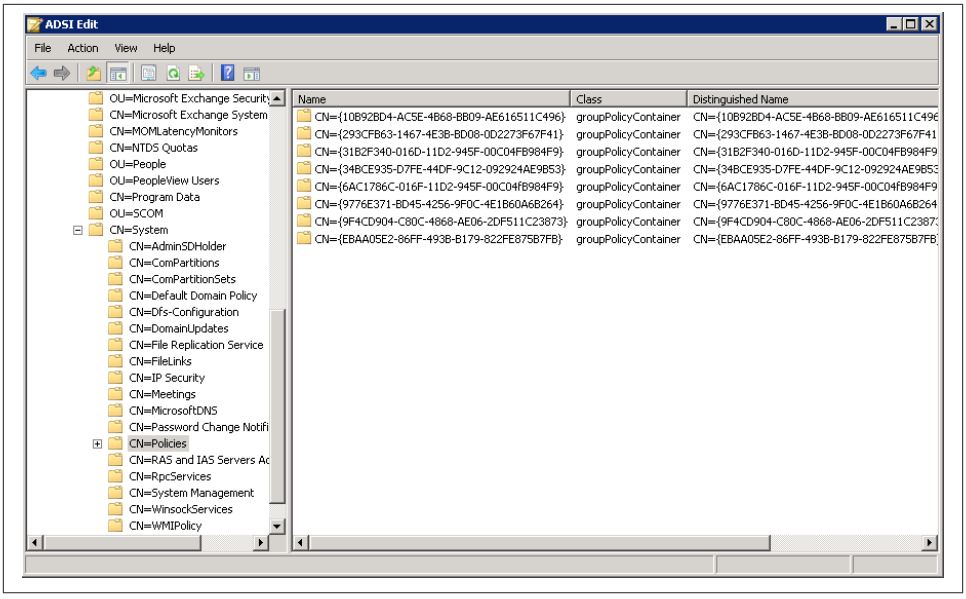


Figure 11-1. GPOs in the Policies container

Group Policy replication

Historically, the SYSVOL container has replicated using the NT File Replication Service (NTFRS), which can perform multimaster replication of a file share. However, NTFRS has never been known for flexibility, or even, in some cases, reliability. Consequently, Windows Server 2008 introduced the ability to replicate the Sysvol folder hierarchy with a new file and folder replication technology called Distributed Files System Replication (DFS-R). DFS-R has substantially enhanced functionality and is much more reliable. It is in your best interest to migrate SYSVOL replication to this service when it becomes possible. For an extensive FAQ on DFS-R, visit *http://bit.ly/11H9Czp*.

Domains that are built starting with the Windows Server 2008 domain functional level automatically use DFS-R to replicate the Sysvol container, but domains that are upgraded or built at a lower functional level will continue to use NTFRS. In order to migrate to DFS-R replication of Sysvol, the domain in question must be at the Windows Server 2008 or newer domain functional level. To move to DFS-R, you will use the *dfsrmig* utility.

The Microsoft Storage Team Blog has an extensive series on the process of migrating Sysvol to DFS-R replication, so rather than try to replicate the detail of this content, we will outline the process and point you directly to the source: *http://bit.ly/13tg2qL*. Each of the subsequent blog posts in the series is linked at the bottom of the page. At a high level, there are four stages of the migration process:

1. Migration start
2. Sysvol prepared state
3. Sysvol redirected state
4. Eliminated state

In the prepared state, new DFS-R objects get created in Active Directory, and each domain controller creates a new *Sysvol_DFSR* folder into which it copies the contents of its SYSVOL folder. The DFS-R service then begins initializing itself. Once all of your domain controllers have reached the prepared state, you can continue to step 3.

When you move to the redirected state, each domain controller updates its *Sysvol_DFSR* folder with any changes made to the active Sysvol share since it entered the prepared state. The domain controller then modifies the Sysvol share so that the *Sysvol_DFSR* folder is shared out as Sysvol. Once all your domain controllers have reached the redirected state, you can continue to step 4.

When you move to the eliminated state, the NTFRS settings for replicating Sysvol are removed, and the old *Sysvol* folder is removed from the filesystem. At this point, your migration to DFS-R replication of Sysvol is complete.

At any time prior to step 4, you can roll back one step or all the way back to step 1. Once you reach the eliminated state, however, you cannot roll back to NTFRS.

How Group Policies Work

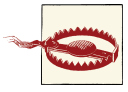
The remainder of this chapter takes an in-depth look at group policy objects, focusing on two areas:

- How GPOs work in Active Directory
- How to manage GPOs

Group policies are very simple to understand, but their usage 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.

GPOs and Active Directory

Any GPO is initially created as a standalone object in Active Directory. Each object can then be linked one or more times to three different types of locations: sites, domains, and organizational units. GPOs for domains and organizational units are held in the domain relating to their application, but creating a GPO for a site stores that GPO in the forest root domain by default; administrators can change this if they wish.



You cannot link group policies to containers. Users and computers that are stored in a container will apply policies linked to the domain or their site, however.

In the normal state of affairs, an administrator would customarily browse to a site, domain, or organizational unit in the GPMC, and then create a GPO and link it to that object. At this point, it appears that you have created a GPO at that location in the tree rather than what really happened, which was that the system created the GPO as a standalone object in the Policies container and then immediately linked it to that container.

To apply a GPO to a set of users or computers, you simply create a GPO and link it to a site, domain, or organizational unit. Then, by default, the user portion of the GPO will apply to all users in the tree, and the computer portion of the GPO will apply to all computers in the tree.

Thus, if we were to create a policy and link it to a domain, all computers and users of that domain would process the policy. If we were to create a policy and link it to an OU, all users and computers in that OU, and all users and computers within OUs beneath that OU (and so on down the tree), would process the policy.

To identify the links on a GPO, you simply look at the Scope tab of the GPO's properties in the Group Policy Management Console. Figure 11-2 shows the results of a scan for the locations in the domain where the Desktop Policy GPO has been linked. In this case, the Desktop Policy has been linked only to the Desktops OU.

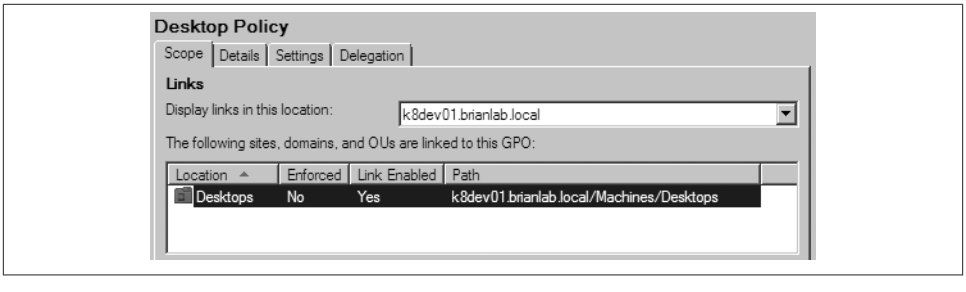


Figure 11-2. Identifying GPO links

We want to make three major points here:

- GPOs can be linked only to sites, domains, and organizational units.
- A single GPO can be linked to multiple locations in the tree.
- GPOs by default affect all of the users and computers in the linked scope.

This generates further questions. If multiple policies apply to different locations in a tree, can multiple GPOs apply to the same location, and if so, what takes precedence? Why would you want to apply one GPO to different parts of the tree? In addition, how can we stop the GPO from applying to the entire set of users and computers in the tree? Let's consider each of these questions to understand policies better.

Prioritizing the Application of Multiple Policies

Let's say that we create and link a GPO for all users in a site to run a logon script that loads an Intranet home page local to that site. Let's also say that we create and link a domain GPO to set the *My Documents* folder location for each user in the domain. Finally, we have two user logon scripts that we need to run in a specific order for specific organizational units in that domain. GPOs are applied in a specific order, commonly called *LSDOU*. Local policies are applied first, and then site policies, then domain policies, and then finally OU policies are applied in the order of the OU hierarchy. This order of application is illustrated in Figure 11-3.

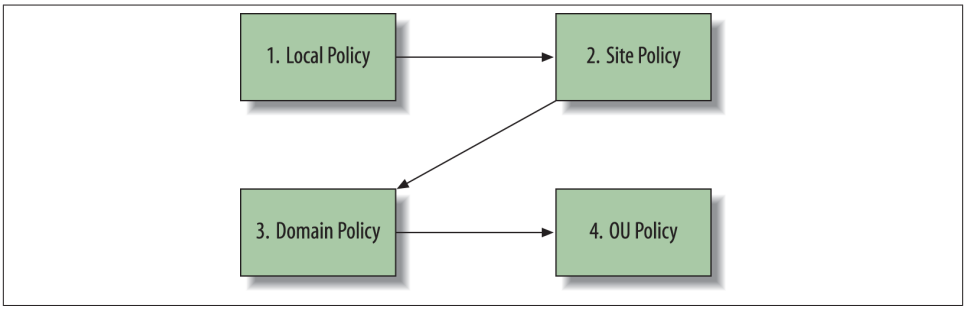


Figure 11-3. Order of policy application

If multiple GPOs are linked to a single site, domain, or organizational unit, the administrator can prioritize the order in which the GPOs from that level are processed. To account for this, the GPOs for the site that the user resides in are applied first, in prioritized order from lowest to highest. No other sites have any influence over this. Then the GPOs for the domain that the user resides in are applied in prioritized order. GPOs applied to parent domains in the domain tree have no influence on objects in domains lower down the tree. Domain trees do not impact GPO application at all.

The OU structure, however, has a significant bearing on what happens with GPOs. GPO links are inherited down an OU tree. So, while a child organizational unit can have its own GPOs linked to it, it also will inherit all of its parent's GPO links. These organizational unit GPOs are applied in order according to the OU hierarchy once the site and domain GPOs have been processed.

There are exceptions. You can block inheritance, force an override, and even define ACLs on objects. We'll cover all these topics later in this section.

For example, Paul Burke's user account has the following distinguished name (see Figure 11-4):

```
cn=PaulBurke,ou=Databases,ou=Gurus,ou=Financial Sector,dc=mycorp,dc=com
```

When Paul logs onto his machine, the site GPOs are applied first, and then the *mycorp.com* domain GPOs. Next come the GPOs on the Financial Sector organizational unit, then the GPOs on the Gurus organizational unit, and finally the GPOs on the Databases organizational unit. From this, it's fairly easy to see how organizational unit hierarchy design has a significant effect on GPO precedence.

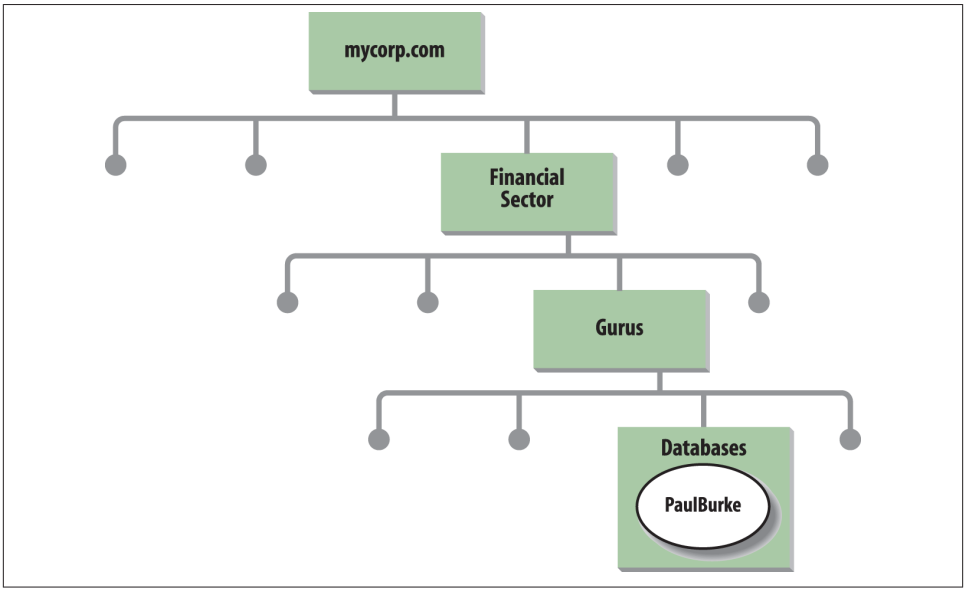
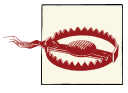


Figure 11-4. Graphical representation of the location of the PaulBurke user

Remember that GPOs have a computer part as well as a user part. When a computer starts up, any site GPOs that have computer settings are applied in prioritized order. These are followed by any domain GPOs with computer settings, and so on down the organizational unit hierarchy until any GPOs on the organizational unit that the computer resides in are applied. During boot up, the user portions of these GPOs are ignored. Later, when a user logs on, the same process applies, this time with the user settings. The computer settings are ignored during user logon.[1]



There are several policies that can only be set on a GPO linked at the domain. These policies are the ones that affect account settings for the domain users, such as the Kerberos, lockout, password aging, and password complexity policies. No matter how hard you try, only the settings at this level will enable you to have different password and lockout policies at an OU level. Any use of the lockout and password policy settings at the OU level will only affect local user accounts on the machines to which the policies apply.

To apply different password or lockout policies to different sets of users, refer to Chapter 12, where we discuss fine grained password policies.

Standard GPO Inheritance Rules in Organizational Units

Any unconfigured settings anywhere in a GPO are ignored, and only configured settings are inherited. There are three possible scenarios:

- A higher-level GPO has a value for a setting, and a lower-level GPO does not.
- A GPO linked to a parent OU has a value for a setting, and a GPO linked to a child OU has a non-conflicting value for the same setting.
- A GPO linked to a parent OU has a value for a setting, and a GPO linked to a child OU has a conflicting value for the same setting.

If a GPO has settings configured for a parent organizational unit and the same policy settings are unconfigured for a child organizational unit, the child inherits the parent's GPO settings. That makes sense.

If a GPO has settings configured for a parent organizational unit that do not conflict with the settings in a GPO configured for a child organizational unit, the child organizational unit inherits the parent GPO settings and applies its own GPOs as well. A good example of this is two logon scripts; these scripts don't conflict, so both are run.

1. This is the default case. There is a setting that you can use to force a different mode of operation. We'll explain this later, when we cover loopback mode.

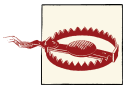
If a GPO has settings configured for a parent organizational unit that conflict with the same settings in another GPO configured for a child organizational unit, the child organizational unit does not inherit those specific GPO settings from the parent organizational unit. The settings in the GPO child policy take priority.

While we only refer to parent organizational units in this section, the parent can be a site, domain, or local policy as well.

Blocking Inheritance and Overriding the Block in Organizational Unit GPOs

It is possible to force the settings of a GPO linked to an OU in the tree to be applied as the final settings for a child.

Blocking inheritance is a fairly simple concept. If you block inheritance to a specific organizational unit, GPOs linked to parent organizational units up the tree are not applied to objects in this specific organizational unit or its children.



Local policies are processed even when Block Inheritance is checked.

Refer back to Figure 11-4. If we decide to block inheritance at the Databases organizational unit, user *PaulBurke* will receive only GPOs directly defined on the Databases organizational unit. If we decide to block inheritance at the Gurus organizational unit, *PaulBurke* will receive only GPOs on the Databases organizational unit and those inherited from the Gurus organizational unit. The organizational unit that you block inheritance at stops any higher-level GPOs from applying to the branch starting at the blocked organizational unit. In fact, we can block inheritance on any of the organizational units within the *mycorp.com* domain. For example, blocking inheritance on the Financial Sector organizational unit makes sense if we want to block domain and site-level GPOs from applying.

This can cause problems if not carefully managed. For example, let's say that you have delegated control over an organizational unit branch to a group of administrators and allowed them access to manipulate GPOs on that branch. You may be applying GPOs to organizational units farther up the hierarchy that you wish this delegated branch to receive. However, your branch administrators have the ability to block inheritance of these parent organizational unit policies of yours. The branch administrators also have the ability to configure a setting that conflicts with one you set in a parent GPO; the branch administrator's child setting will take precedence in conflicts.

You can block inheritance by right-clicking a domain or organizational unit in the GPMC and selecting Block Inheritance, as shown in Figure 11-5.

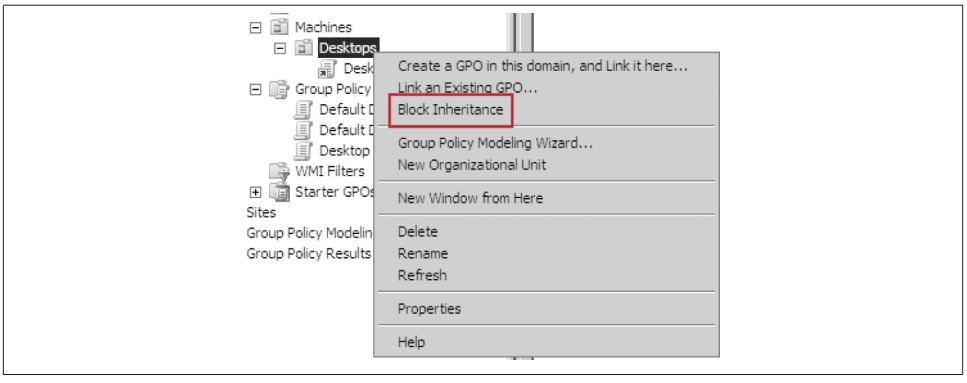


Figure 11-5. Blocking inheritance with the GPMC

To prevent this, you can check the Enforced option on an individual GPO link. This allows administrators to force GPOs to be inherited by all children of an organizational unit. However, it has one further effect: it prevents GPO settings in child organizational units from overriding conflicting settings in a parent OU. The Enforced option is available by right-clicking on a group policy link in the GPMC and selecting Enforced, as shown in Figure 11-6.

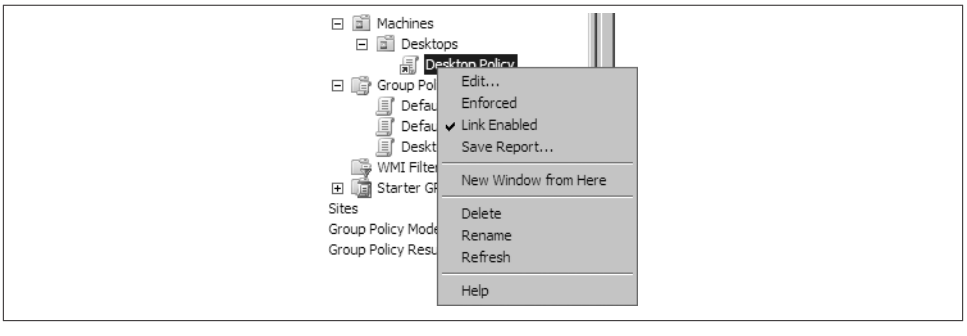


Figure 11-6. Group policy link context menu

The Enforced setting is configured on a per-group policy link basis. You cannot set a given group policy object to be globally enforced through the domain.

Let's say that we change a registry setting using a GPO on the Financial Sector organizational unit. Unfortunately, another administrator then sets the same registry setting (among many others) to a conflicting value on the Gurus organizational unit and also blocks inheritance at the Databases organizational unit. By default, the registry setting will be correctly applied only to the Financial Sector organizational unit, as the Gurus organizational unit receives the different setting (child overrides parent on conflicts due to inheritance rules), and the Databases organizational unit doesn't inherit either policy. To fix both problems, we could set the original Financial Sector organizational unit GPO link to Enforced. This would prevent the specific setting on the GPO on the Gurus organizational unit from modifying it without affecting any of the other GPO settings. Our GPO would also be forced down past the Block Inheritance set up at the Databases organizational unit.

If you perceive a trust issue in who can modify a given policy, it is suggested that you consider setting up an ACL on that policy to restrict the abilities of others to edit that GPO, leaving just a core group of administrators with the relevant permissions. This will ensure that the GPO is not changed without the knowledge of the core group. Of course, the fundamental issue is allowing anyone to have access to make any changes that you do not implicitly trust. Trying to secure bits and pieces from administrators you don't trust is a losing battle. If you do not trust someone to be an administrator, he should not be one.

While it's important to be aware of the Block Inheritance and Enforced settings, we generally don't recommend that you make use of them in your Group Policy deployment unless you absolutely have to. Both of these settings modify the default expected behavior, and as a result they make troubleshooting more complicated. If you do take advantage of either setting, be sure to document your reasons so that administrators aren't left guessing in the future.

Summary

- If Block Inheritance has been checked for a child-level OU and Enforced has not been checked for any parent GPOs, the child GPO will not inherit any policies from any parent GPOs farther up the hierarchy.
- If Enforced has been checked for a parent-level GPO, the child-level OU will inherit all of the enforced GPO's configured policies, even if those policies conflict with the child's policies and even if Block Inheritance has been set for the child.

When Policies Apply

We've already said that the computer portion of a GPO applies during boot up and the user portion of a GPO applies during logon. However, that isn't the only time that a policy can apply. The policies also can be set to refresh periodically after a certain time interval. How often this occurs and what conditions are attached to this refresh operation are specified under the `System\Group Policy` key under the Administrative Templates section of the Computer section of a GPO.

Set the refresh value to 0 to have the policy continually apply every seven seconds. This could be very useful for a test environment, but obviously not for a production environment.

If you need to trigger a group policy refresh outside of the normal schedule, you can use the *gpupdate* tool included with Windows. If any policies were modified that can only be applied at computer startup or user logon, *gpupdate* will ask if you want to reboot or log off. Windows also includes a tool called *gpresult* to display details on what policies are applied.

Windows 8 clients include the Invoke-GPUpdate PowerShell cmdlet, which has the same functionality as *gpupdate*.

Refreshing is very useful for users who do not shut down their computers or log off from the system for days. In that case, GPOs apply in the normal way, but at very irregular intervals over long periods at a time. Setting up policy refresh means that you can apply those settings to such users at whatever interval you decide.

Group Policy Refresh Frequency

By default, Windows workstations and member servers refresh their policies every 90 minutes, and domain controllers refresh their policies every 5 minutes. In order to avoid having all machines retrieving their policies at once from the domain controllers, there is a random offset interval added to the refresh period on every machine. This offset interval, by default, is a random value up to 30 minutes on workstations and member servers, and 0 minutes on domain controllers. Both the refresh interval and the offset interval are configurable. To modify these settings, you can modify the registry of a computer or you can edit a group policy object that applies to the computer. The GPO settings are in the Group Policy section of the System portion of the Administrative Templates section.

The registry entries for computer policies are under the `HKLM\Software\Policies\Microsoft\Windows\System` key. The specific values are listed in Table 11-1.

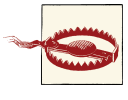
Table 11-1. Computer group policy refresh registry values

Registry value name	Description
`GroupPolicyRefreshTime`	*Description:* refresh interval for domain members *Data type:* `REG_DWORD` *Valid range:* 0 – 64,800 minutes
`GroupPolicyRefreshTimeOffset`	*Description:* offset interval for domain members *Data type:* `REG_DWORD` *Valid range:* 0 – 1,440 minutes
`GroupPolicyRefreshTimeDC`	*Description:* refresh interval for domain controllers *Data type:* `REG_DWORD` *Valid range:* 0 – 64,800 minutes
`GroupPolicyRefreshTimeOffsetDC`	*Description:* offset interval for domain controllers *Data type:* `REG_DWORD` *Valid range:* 0 – 1,440 minutes

The registry entries for user policies are under the `HKCU\Software\Policies\Microsoft\Windows\System` key. The specific values are listed in Table 11-2.

Table 11-2. User group policy refresh registry values

Registry value name	Description
`GroupPolicyRefreshTime`	*Description:* refresh interval for users *Data type:* `REG_DWORD` *Valid range:* 0 – 64,800 minutes
`GroupPolicyRefreshTimeOffset`	*Description:* offset interval for users *Data type:* `REG_DWORD` *Valid range:* 0 – 1,440 minutes.



While we provide the registry values here that Group Policy will use, we do not recommend that you edit these registry values manually. Always use a group policy object to configure these settings instead.

Combating Slowdown Due to Group Policy

Introducing Group Policy into your environment will affect computer startup and user logon times to a degree. Exactly what degree varies from environment to environment, and you will need to test in yours to come up with a representative figure. We do not recommend foregoing Group Policy in an effort to speed up your startup and logon times, but we do recommend being frugal when planning the number of policies that will apply to a given user or computer. There are some things you can take into consideration with regard to speed when planning your Group Policy deployment.

Limiting the number of GPOs that apply

Microsoft has its own take on designing your Active Directory infrastructure for GPOs. The recommendation is that you should not have organizational unit structures more than 10 deep, so that applying policies doesnot take too much time during logon. Note that the issue isn't the number of OUs; it is the number of policies linked to the OUs. So, if you have 5 policies linked, you'll see the same performance hit if these are linked over 3 OU levels or 10 OU levels.

Limiting cross-domain linking

It is possible for an administrator of one domain to create a GPO and for it to be applied to a site, domain, or organizational unit in another domain. For example, if the administrator of *child.mycorp.com* is given access to centralized setup GPOs within *mycorp.com*, she can link the *mycorp.com* GPOs to domains or organizational units in the *child.mycorp.com* domain.

While this is a feasible setup, it is generally not recommended. In order for clients in the *child.mycorp.com* domain to apply policies from *mycorp.com*, they must contact a domain controller in *mycorp.com* so they can access the Sysvol share in *mycorp.com*. This will slow down logon and startup times, especially if the connectivity is slow. In lieu of linking policies between domains, we generally recommend that the policies be physically duplicated in each domain. Duplicating policies by way of import/export is discussed later in this chapter.

For more information about GPOs and WAN links, see the upcoming sidebar "How GPOs Work Across Slow Links" on page 300.

Limiting use of site policies

Applying site policies has the same limitations as cross-domain linking if you are running in a multidomain environment. By default, site-based policies are stored on the domain controllers in the root domain of the forest. For clients to apply these policies, they must contact a domain controller in the root domain to retrieve the policy.

We generally recommend that organizations pursue alternatives to site-based policies. Site-based policies are often confusing for administrators to manage and troubleshoot, as the administrator must remember to check the site, in addition to the domain and organizational units, for policies. Furthermore, site-based policies often require traversing one or more WAN links in order to retrieve the policies.

How GPOs Work Across Slow Links

GPOs and even user profiles can still work across slow links, and a lot of the configuration is left in the hands of the administrator. Administrators can specify what speed is used in the definition of a slow link. For computers and users, the following policy areas need looking at:

- Policies→Computer Configuration→Administrative Templates→System→Group Policy→Group Policy slow link detection
- Policies→User Configuration→Administrative Templates→System→Group Policy→Group Policy slow link detection

In both cases, the default setting is 500 KBps, but administrators can set any KBps connection speed time that they wish. This speed is used against a slow-link-detection algorithm; if the speed is above the value, the link is fast, while a speed below the value indicates a slow link.

This is the algorithm in pseudocode prior to Windows Vista/Windows Server 2008:

```
Ping server with 0KB of data :
   Receive response#1 as a time in milliseconds (ms)
If response#1 < 10ms Then
   Exit as this is a fast link
Else
   Ping server with 2KB of data :
      Receive response#2 as a time in milliseconds
   Calculate Total-speed as response#2-response#1
End If
Ping server with 0KB of data :
   Receive response#1 as a time in milliseconds
If response#1 < 10ms Then
   Exit as this is a fast link
Else
   Ping server with 2KB of data :
      Receive response#2 as a time in milliseconds
   Calculate Total-speed as Total-speed + (response#2-response#1)
End If
Ping server with 0KB of data :
   Receive response#1 as a time in milliseconds
If response#1 < 10ms Then
   Exit as this is a fast link
Else
   Ping server with 2KB of data :
      Receive response#2 as a time in milliseconds
   Calculate Total-speed as Total-speed + (response#2-response#1)
End If
 'Average the total speed of (response#2-response#1)
Difference-in-milliseconds = Total-Speed/3
```

```
'If we know 2KB (16,384 bits) was moved in a certain number
'of milliseconds, then we need to calculate the number
'of bits moved per second (not per ms)
Bits-per-second-value = (16384 * 1000/Difference-in-milliseconds)
'Eight bits is a byte, so calculate bytes/second
bps-value = (Bits-per-second-value * 8)
'Calculate kilobytes/second to compare against GPO value
Kbps-value = bps-value / 1024
```

Windows Vista/Windows Server 2008 and newer clients depend upon the Network Location Awareness service to determine if a link is slow.

The following GPOs are applied across slow links:

- When a user logs in using the "Logon using dial-up connection" checkbox on the logon screen, user policies are applied.
- When the computer is a member of the same domain as the remote access server (RAS) or is in a domain with a trust relationship to the one the RAS server is in, both sets of policies are applied.

GPOs are not applied:

- When the logon is done using cached credentials
- To computers that are members of a different domain

In all of these cases, security settings (i.e., IP security, EFS recovery, etc.) and Administrative Template settings are the only ones to be applied by default; folder redirection, disk quota, script, and software installation policies are not applied. You can't turn off registry settings you have to apply. You can, however, alter the default state of any of the others, including the security settings, using the relevant sections of those GPOs.

Use simple queries in WMI filters

If you have a Windows Management Instrumentation (WMI) filter applied to a GPO, a WMI query will be run before the GPO is applied to a user or computer. If the WMI query is very complex, it could significantly impact the time it takes to process the GPO. If you have multiple GPOs that contain a WMI filter, you need to pay special attention to the impact those queries will have. If you are using WMI filters, check out the free WMI Filter Validation Utility. This utility will allow you to test WMI filters as well as see how long they take to apply.

Security Filtering and Group Policy Objects

As each GPO is an object in Active Directory and all objects have access control lists (ACLs) associated with them, it follows that it must be possible to allow and deny access to a GPO using that ACL. With ACLs, it is possible to allow and deny access to apply

GPOs based on security group membership. It is also possible to go to an even finer-grained level of detail and set access controls for an individual computer or user. Figure 11-7 shows that only one computer, *XPCLNT01*, can apply the policy pictured.

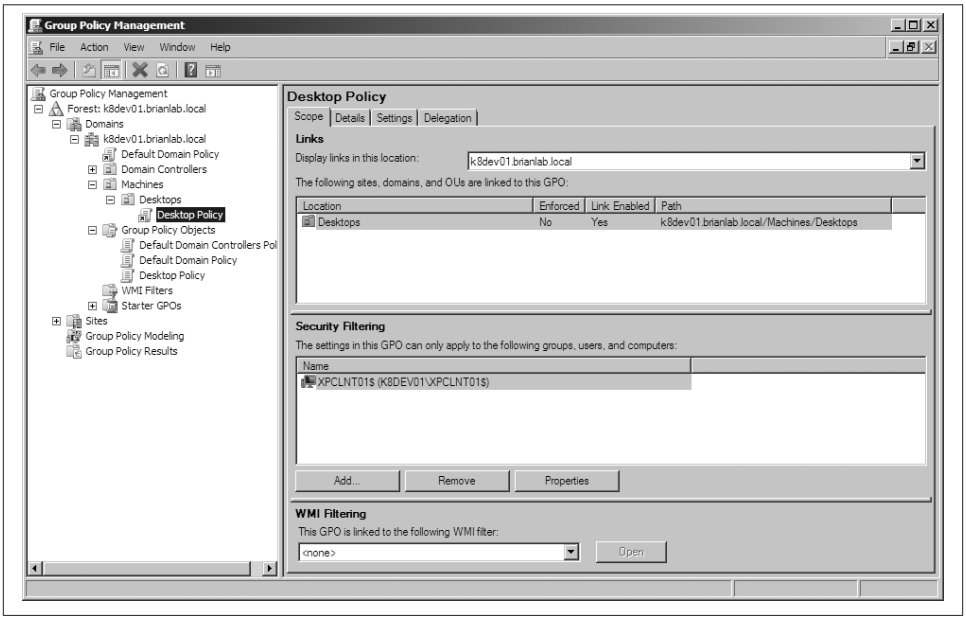


Figure 11-7. Security filtering with the GPMC

This is a significant feature of GPOs, and one that you can use heavily to your advantage. Let's take a simple example, in which we need to ensure that users in the Finance Department receive a more restrictive screensaver timeout. The users in the Finance Department are in the People OU, along with all of the other company employees. In order to ensure that the screensaver timeout policy only applies to users in the Finance Department, we take the following steps:

1. Create a Screensaver Timeout GPO.
2. Configure the "Screensaver Timeout" setting (*User Configuration\Policies\Administrative Templates\Control Panel\Personalization*).
3. Remove Authenticated Users from the GPO's security filter and add the Finance Users group.
4. Link the GPO to the People OU.

Once this is complete, only users who are members of the Finance Users group will be able to apply the Screensaver Timeout GPO. All other users in the People OU will ignore the policy.

Loopback Merge Mode and Loopback Replace Mode

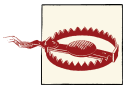
Up until now, we've only been able to apply the User Configuration section of a GPO when users were located somewhere under where the GPO was linked. From time to time, scenarios arise where you will want to apply User Configuration policies based on what machine the user is logging into rather than where the user is located in the domain's OU structure. The solution to this is a Group Policy feature known as *loopback mode*.

There are two scenarios where loopback mode is very commonly used. The first is in the case of kiosk machines. If your company has computers available for employee use in common areas, such as building lobbies, you may only want to allow users to access Internet Explorer from the kiosk machine and prevent access to other programs. Another scenario is the case of remote desktop servers (RDSs) and Citrix deployments. In these cases, you will generally want users of the RDS server to receive a controlled desktop environment when they connect, but you will not want to apply that level of control to the users' desktop computers.

Loopback mode is a computer policy that you must enable. You can find the settings under *Computer Configuration→Policies→System→Group Policy→User Group Policy loopback processing mode*. If you open that setting, you get the dialog box shown in Figure 11-8, which allows you to switch between the two modes of loopback operation: merge mode and replace mode.

At this point, you are probably wondering what the difference is between the merge and replace modes. In *merge mode*, when a user logs onto a machine, his user policies are first applied in the normal manner, as shown in Figure 11-3. Once the user policies have been applied, the User Configuration sections of any GPOs that apply to the computer are then applied. Any settings that conflict (that is, they are defined in both the normal user policies and the policies applying to the computer) are overridden by the settings in the policies applying to the computer. In *replace mode*, by contrast, the GPOs that would normally apply to the user are ignored during logon. Instead, the User Configuration sections of any GPOs that apply to the computer are applied.

When you are planning to take advantage of loopback mode, it is critical to remember that loopback mode is a global setting for a computer, not a per-GPO setting. This means that if loopback mode is enabled in *any* GPO that applies to a computer, loopback processing will be enabled for all users that log onto that computer, and further, the User Configuration settings in all of the GPOs that apply to that computer will be applied.



The idea that loopback mode is enabled on a per-GPO basis is a common misconception amongst Active Directory administrators. Make sure you are aware of this behavior before enabling loopback mode!

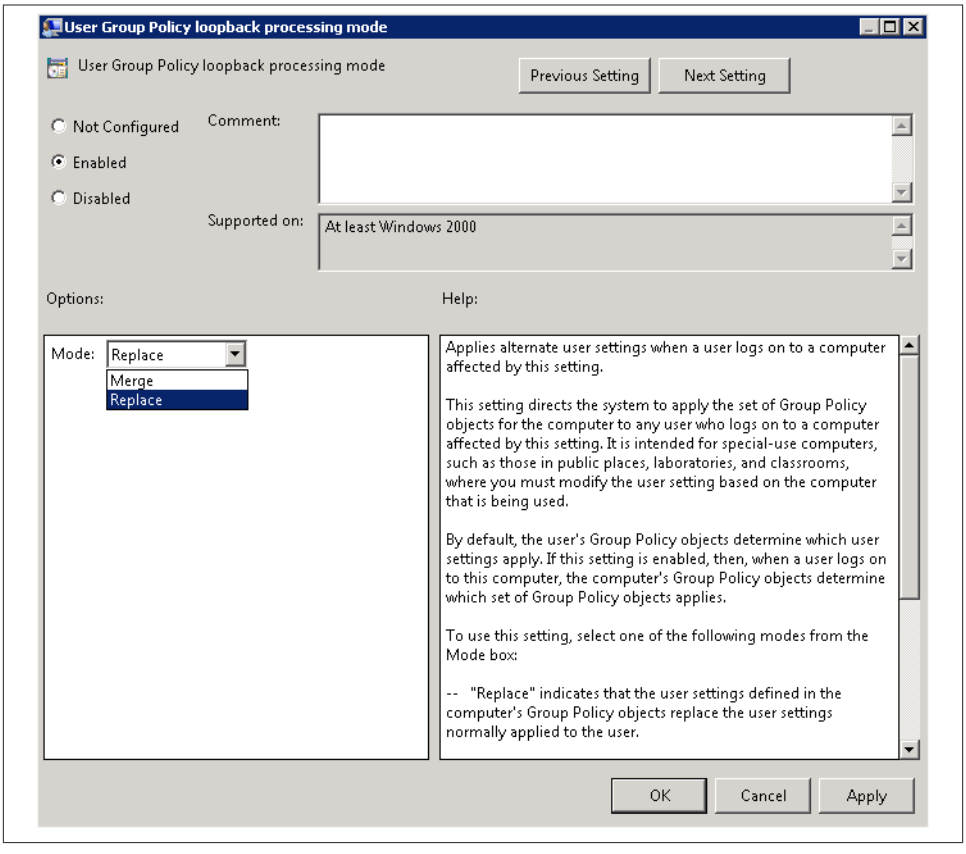


Figure 11-8. Setting loopback mode

Summarizing Group Policy Application

Consider the organization in Figure 11-9. The organization uses a domain called *mycorp.com* that spans two sites, Main-Site and Second-Site. Marketing computers exist in Main-Site, and finance computers in Second-Site. Policy A applies to Main-Site only, policy B applies to the entire domain, and policies C, D, E, and F apply to the organizational units as indicated. Policy G applies to Second-Site.

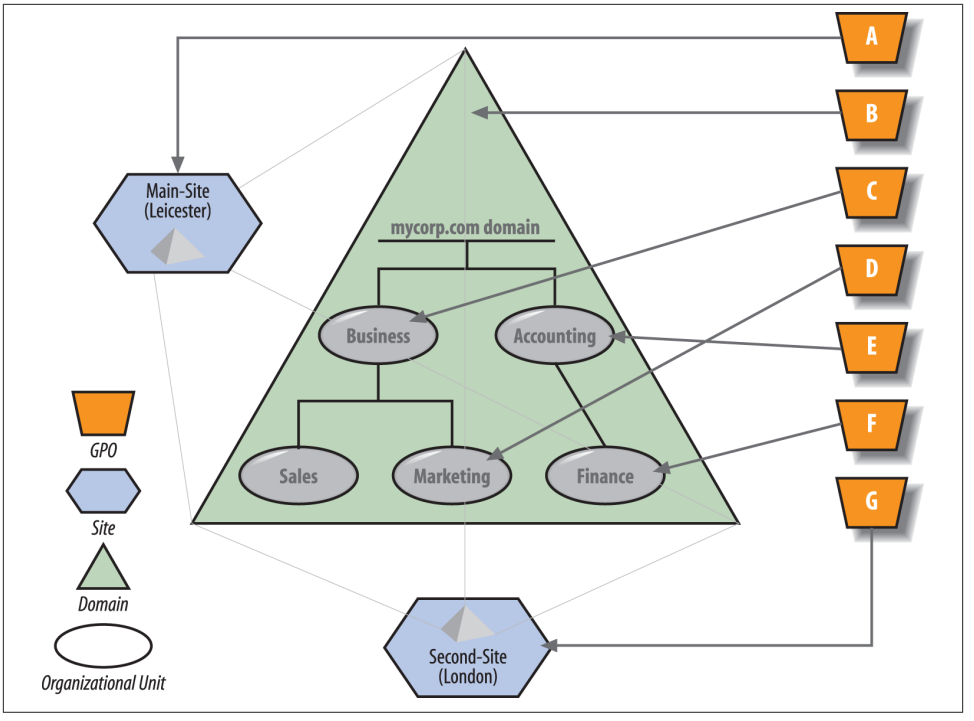


Figure 11-9. Sample Group Policy implementation

Table 11-3 summarizes the application of policies with and without loopback mode processing. When loopback is not turned on, the only real difference comes from the site policies (A or G) that are applied. When you turn merge mode on for all the GPOs, it becomes more obvious what will happen. In each case, the policies relating to the user are applied first, in order, followed by the entire set of policy items that would apply to a user residing in the computer location. Take the example of a finance user logging on at a marketing computer in the main site:

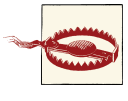
1. The user portion of the site policy that the user is logging on from (A) is applied.
2. The user portion of the domain policy (B) is applied.
3. The user portion of the Accounting organizational unit policy (E) is applied.
4. The user portion of the Finance organizational unit policy (F) is applied.
5. The user portion of the site policy (A) is applied again.
6. The user portion of the domain policy (B) is applied again.
7. The user portion of the Business organizational unit policy (C) is applied.
8. Finally, the user portion of the Marketing organizational unit policy (D) is applied.

Table 11-3. Resultant set of policies for available loopback modes

Loopback mode in use	OU in which user resides	Where computer resides	Resultant set of policies
No	Marketing	`OU=Marketing` (main site)	ABCD
No	Finance	`OU=Finance` (second site)	GBEF
No	Marketing	`OU=Finance` (second site)	GBCD
No	Finance	`OU=Marketing` (main site)	ABEF
Merge	Marketing	`OU=Marketing` (main site)	ABCDABCD
Merge	Finance	`OU=Finance` (second site)	GBEFGBEF
Merge	Marketing	`OU=Finance` (second site)	GBCDGBEF
Merge	Finance	`OU=Marketing` (main site)	ABEFABCD
Replace	Marketing	`OU=Marketing` (main site)	ABCD
Replace	Finance	`OU=Finance` (second site)	GBEF
Replace	Marketing	`OU=Finance` (second site)	GBEF
Replace	Finance	`OU=Marketing` (main site)	ABCD

Remember that policies applied later can override the policies that were applied earlier, so the user portions of the policies applying to the location of the computer will always override previous policies if there is a conflict. With policy order ABEFABCD, D can override C, which can override B, which can override A, which can override F, and so on. Also, in all these cases, if any of the computer GPOs do not have any defined settings in the user portion, those policies are ignored.

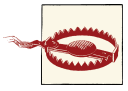
Loopback replace mode is used when the user portions of the GPOs that apply to a computer are to be the only ones set. For the finance user logging onto a marketing computer in the main site, the only policies that get applied to that user are ABCD, the user portions of the GPOs that apply to the marketing computer.



Administrators must be aware that loopback mode can impact logon times, especially when using loopback in its merge mode.

WMI Filtering

Windows Management Interface filtering enables you to associate a WMI query with a GPO, which will run for each user and computer to which the GPO applies. A WMI filter can utilize any WMI-based information that is accessible from the client's machine, including computer hardware and configuration, user profile, and environment settings. This presents a lot of options for targeting GPOs to clients that have certain properties.



WMI filters will only run on Windows XP and newer operating systems. If a policy with a WMI filter attached applies to a Windows 2000 computer, the WMI filter will be ignored.

For example, let's say you want to apply a certain GPO if a client is accessing your network over a virtual private network (VPN) connection. Depending on which VPN software the client is running, your WMI query could check for the existence of a process or service, or even an IP address range. If the query returns true, the GPO will be applied; if it returns false, it will not be applied.

Group Policy

That's a lot of information on GPOs. Let's summarize what we've covered about the workings of GPOs so far:

- GPOs exist in a split state. The configuration data for the GPO, known in shorthand form as *GPC data*, is held in the AD object itself. The template files and settings that tell the GPO what its capabilities are, known in shorthand form as *GPT data*, are stored in the SYSVOL.
- Individual GPOs can be linked to multiple sites, domains, and organizational units in Active Directory, as required.
- GPOs can contain policies that apply to both computers and users in an OU. The default operation of a GPO on a container is to apply the computer portion of the GPO to every computer in that container during boot up and to apply the user portion of the GPO to every user in that container during logon. GPOs can also be set to refresh periodically.
- Multiple GPOs linked to a particular container in Active Directory will be applied in a strict order according to a series of priorities. The default prioritized order corresponds to the exact order in which the GPOs were linked to the container. Administrators can adjust the priorities as required.
- While GPOs exist only in a domain environment due to their dependence on Active Directory, individual domain or workgroup computers can have local policies defined for them.
- GPOs are inherited down the organizational unit hierarchy by default. This inheritance can be blocked using the properties of an OU, domain, or site. Administrators can also set a policy to be *enforced*. This allows a policy to override all lower settings and bypass any blocks.
- Loopback mode allows the administrator to specify that user settings can be overridden on a per-machine basis. Effectively, this means that the user parts of policies that normally apply only to computers are applied to the users as well as (merge mode) or instead of (replace mode) the existing user policies.

- WMI filtering allows you to configure a WMI query that can be used as an additional criterion to determine whether a GPO should be applied. If the filter evaluates to true, the GPO will continue to be processed; if it evaluates to false, the GPO will not be processed. This is a powerful feature because you have a vast amount of WMI data available to determine whether GPOs should be applied.
- A number of things can slow down processing on a client, including attempting to process many policies one after the other. Use of loopback mode, especially in merge mode, can significantly impact performance. Attempting to apply GPOs across domains can also lead to slowdowns, depending on the network speed between the domains. Finally, complex queries in WMI filters can have a negative impact on GPO processing.
- Policies are applied in a strict order known as LSDOU. This notation indicates that local policies are applied first, followed by site GPOs, domain GPOs, and finally any organizational unit GPOs hierarchically down the tree. At each point, the policies are applied in prioritized order if multiple policies exist at a location.
- When policies are to be applied to a client, the system identifies the entire list of policies to be applied before actually applying them in order. This is to determine whether any blocking, overriding, or loopback settings have been put in place that could alter the order or application of the policies.
- ACLs can be used to limit the application of GPOs to certain individual users or computers or groups of users or computers. Specifically setting up the ACLs on a GPO to deny or allow access means that you can tailor the impact of a policy from the normal method of applying the GPO to all users or computers in a container.
- Finally, both user profiles and policies can be applied across a slow link, but the speed that the system uses to determine whether a link is slow is configurable by the administrator within an individual GPO. In addition, while security settings and administrative templates normally are applied by default, the exact settings that will apply across a slow link when one is detected are configurable by the administrator. The only exception is that administrative templates will always be applied; the administrator has no control over this.

Managing Group Policies

As we discussed earlier, the primary tool you'll be using with Group Policy is the Group Policy Management Console (GPMC). The GPMC provides an interface to view and edit policies, understand where policies are applied in your organization, and troubleshoot issues. You can use the Group Policy Modeling and Group Policy Results functions in the GPMC to understand how policies will apply to users and computers as well as to look at actual results on an end user machine.

Using the Group Policy Management Console

The GPMC is a one-stop shop for all your GPO management needs. You can browse a forest and see where GPOs are applied, you can create and link GPOs, and you can import and export, back up and restore, delegate control, and view RSoP reports, all from the GPMC.

When you open the GPMC, it will default to showing the domain in which your user account is located. If you need to edit GPOs in another forest, right-click the Group Policy Management node in the tree and click Add Forest. If you need to edit GPOs in another domain in the forest, right-click the Domains node and click Show Domains.

When you are editing GPOs, the Group Policy tools connect to and use the PDC emulator FSMO role owner by default. This ensures that multiple copies of the GPME on different machines are all focused on the same DC. This behavior may be overridden if the PDC is unavailable. If this happens, the GPMC will display an error dialog and the administrator may select an alternate DC to use. An administrator can also manually target a different domain controller in the GPMC.

If GPOs are edited on multiple DCs, this can lead to inconsistencies because the last person to write to the GPO wins. For this reason, you should avoid editing GPOs on multiple DCs.

Refer back to Figure 11-7 to see what the GPMC looks like when viewing a GPO. As you can see in the left pane, you can browse through the domains in a forest down to specific organizational units. If you right-click on an organizational unit, you'll get many of the same options as shown in Figure 11-6.

In Figure 11-7, the Desktops organizational unit has been expanded to show that the Desktop Policy GPO has been linked to it (i.e., icon with a shortcut/arrow symbol). A virtual Group Policy Objects container is expanded, which shows all of the GPOs that have been created in the domain. There is also a virtual WMI Filters container that holds any WMI filter objects that have been created. Note that the Group Policy Objects, WMI Filters, and Starter GPOs containers are virtual. By virtual, we mean that the GPMC is simply displaying these objects in dedicated containers to make the GPMC more useable. In reality, these objects are not stored in dedicated containers in Active Directory.

You can also browse the GPOs that have been linked to a site by right-clicking on the Sites container and selecting Show Sites. You have the option of which sites to display.

If we take a look at Figure 11-7 again, we can see that Desktop Policy was selected in the left pane, and several options and settings are displayed in the right pane. The following list is a summary of what's available on each tab:

Scope
: Under the Scope tab, you can view the domains, sites, or organizational units that have been linked to the GPO and delete them if necessary. You can also view what security groups the GPO applies to, and add and remove groups to and from the list. Finally, you can set the WMI filter that should be associated with the GPO.

Details
: The Details tab contains information about who created the GPO, the dates on which it was created and last modified, and the current user version and computer version. The only thing that can be set on this tab is beside GPO Status, which defines whether the user and/or computer settings are enabled.

Settings
: The Settings tab provides a nice shortcut to view which settings have been configured in a GPO. Unlike in the GPME, where you have to drill down through each folder to determine which settings have been configured, you can view the Settings tab for a GPO in the GPMC to see only the options that have been set.

Delegation
: The Delegation tab is similar to the Delegation of Control Wizard, but it's specifically for GPOs. We'll cover this screen in more detail later in the chapter.

Using the Group Policy Management Editor

While the GPMC is used for creating and managing the application of GPOs, actually editing the contents of a GPO requires another tool, known as the Group Policy Management Editor (GPME). The GPME can be launched from the GPMC for any GPO by right-clicking and selecting Edit, as shown in Figure 11-10. Inside the GPME, settings are divided under Computer Configuration and User Configuration, and then again under Policies and Preferences. Policies are the settings you're used to if you've worked with Group Policy before. Preferences might be new to you—we'll take a look at the group policy preferences functionality that Microsoft acquired in 2006 in the next section.

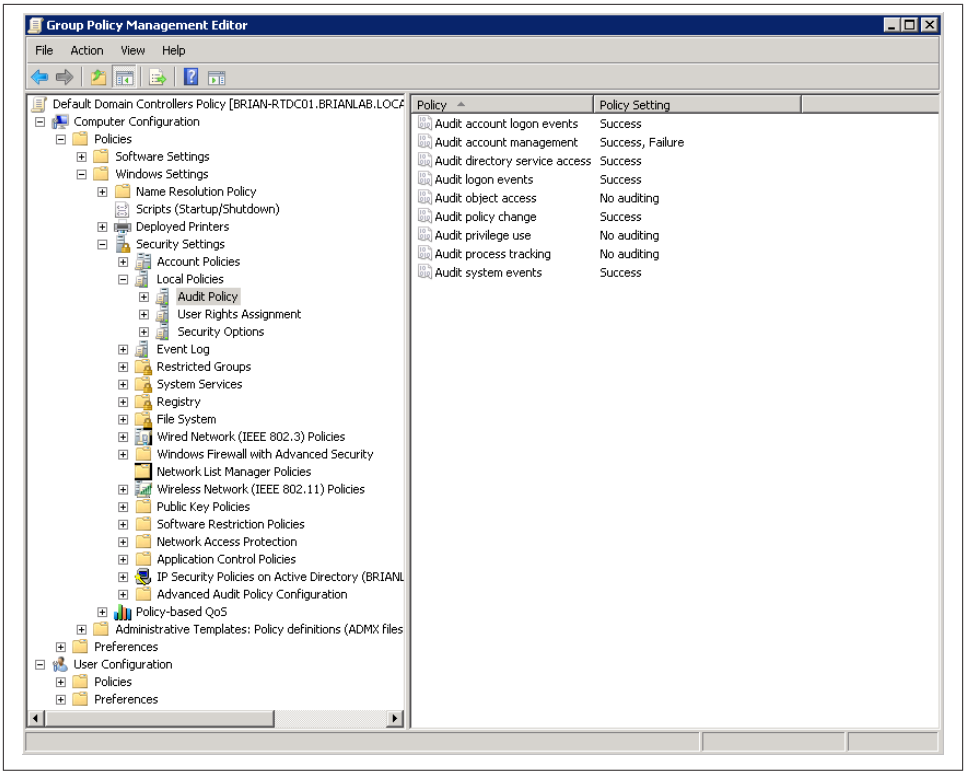


Figure 11-10. Group Policy Management Editor

We won't spend too much time on the Policies section of the GPME, as each setting is a bit different in terms of how it is configured, but you should take some time to click through and become familiar with the interface.

You'll find that the bulk of the policies you might want to apply are located under the Administrative Templates section of the interface. If you're not sure where to find what you're looking for, there are a couple of resources at your disposal. First, you can right-click Administrative Templates and select Filter Options. From here you can configure a keyword search of all the settings available, as well as filter based on what products or versions of Windows they apply to. In Figure 11-11, we're looking for settings that contain the word "screensaver" that have been configured in the policy. Once you're done searching, right-click again and uncheck Filter On, or click the filter button in the GPME toolbar to remove the filter.

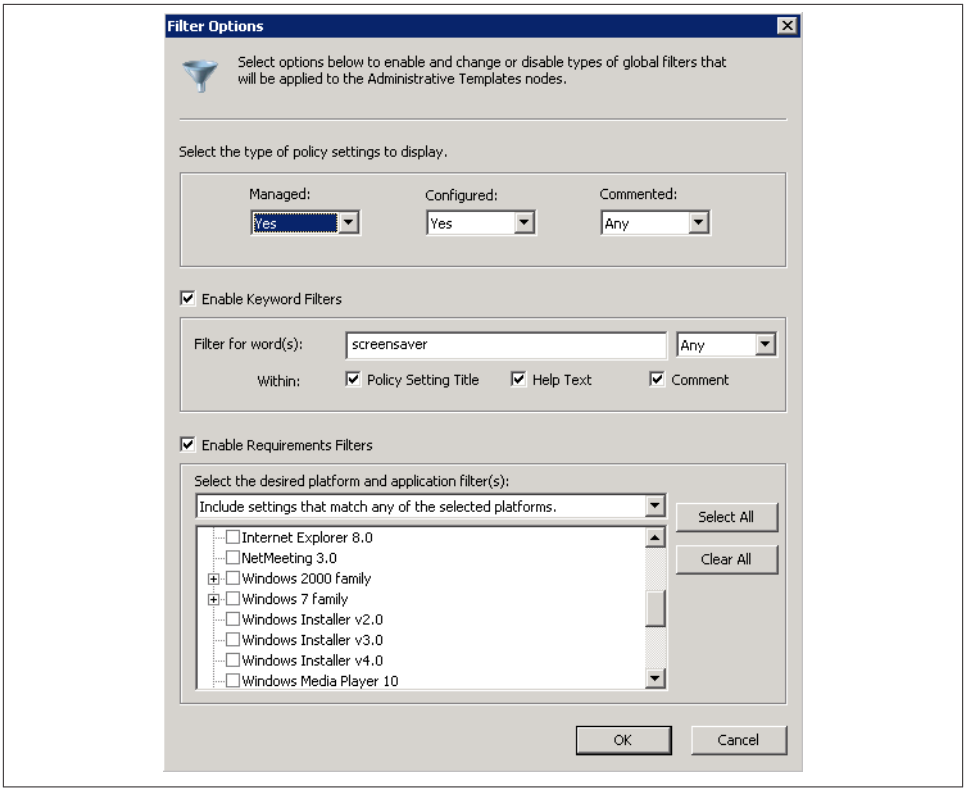


Figure 11-11. Administrative Templates filtering options

If you're simply looking for a specific setting in order to configure it or understand what it does, you might find the free web-based Group Policy Search tool at *http://gps.cloudapp.net* useful. This tool allows you to filter, do more complex keyword searches, and review all of the data from the ADMX files that is also displayed in the GPME.

One of the relatively new features of the GPME is the ability to attach comments to Administrative Template settings, as shown in Figure 11-12. Previously you had to document your group policy choices in a separate tool (or, as is the case at many companies, not at all). With the comment functionality, your group policies can become self-documenting. You can leave a reminder for yourself about why a setting is in place, and ensure that future administrators aren't left guessing.

You can also apply comments to individual GPOs by right-clicking the group policy at the top of the tree in the GPME, clicking Properties, and selecting the Comment tab.

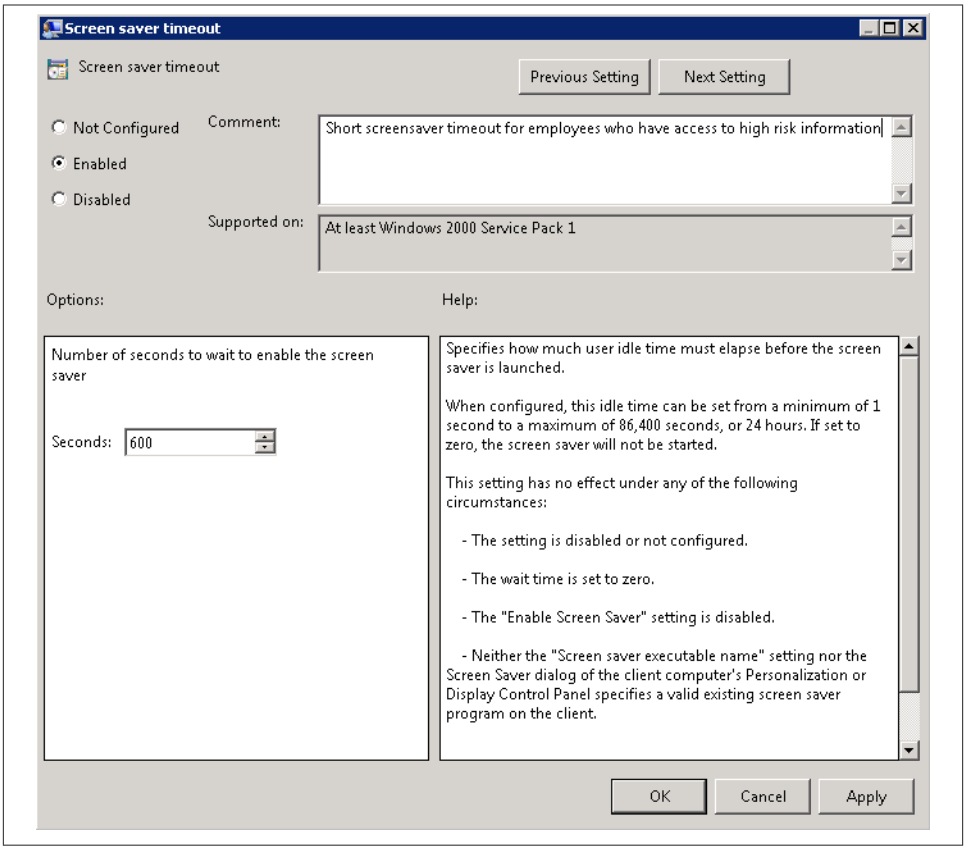
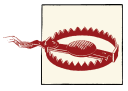


Figure 11-12. Adding comments in the GPME

Group Policy Preferences

Microsoft acquired a company called DesktopStandard in 2006, and as part of that deal it acquired a product called *PolicyMaker*. PolicyMaker morphed into group policy preferences and started shipping with Windows with Windows Server 2008. With group policy preferences, you can graphically configure many components of the desktop experience that previously required complex startup and logon scripts. Some of the highlights of group policy preferences functionality include:

- Drive mappings
- Shortcuts
- Printers
- Registry settings
- Power management



Some of the options available through group policy preferences enable you to store a username and password (such as for a drive mapping or a scheduled task). These passwords are not encrypted in a secure manner. The Group Policy team at Microsoft discusses this on its blog.

Take a look under Preferences in the GPME to explore all of the different options you can configure with group policy preferences. Microsoft has also published a whitepaper about group policy preferences that you can download from *http://bit.ly/14lRcbp*.

While group policy preferences started shipping with Windows Server 2008, it has no dependency on the version of Active Directory you're using. As long as you're using the Group Policy tools from Windows Server 2008 or newer, you can take advantage of group policy preferences.

Group policy preferences work a bit differently than normal group policyies in terms of their application. Namely, the settings are only applied during startup and logon. Background refreshes of GPOs will not apply the preferences settings. Because of this, preference settings are not enforced in the same way as normal policies. Users can often work around or undo the changes that are made by group policy preferences.

With most Group Policy settings, when you remove a setting from a policy or a machine from the scope of a policy, those settings will be removed. Group policy preferences, on the other hand, do not behave this way. If, for example, you map a drive with group policy preferences, you will need to create a preference item to un-map that drive rather than simply deleting the drive mapping preference. This behavior is controlled with the Action option in the properties of each preference item, as shown in Figure 11-13.

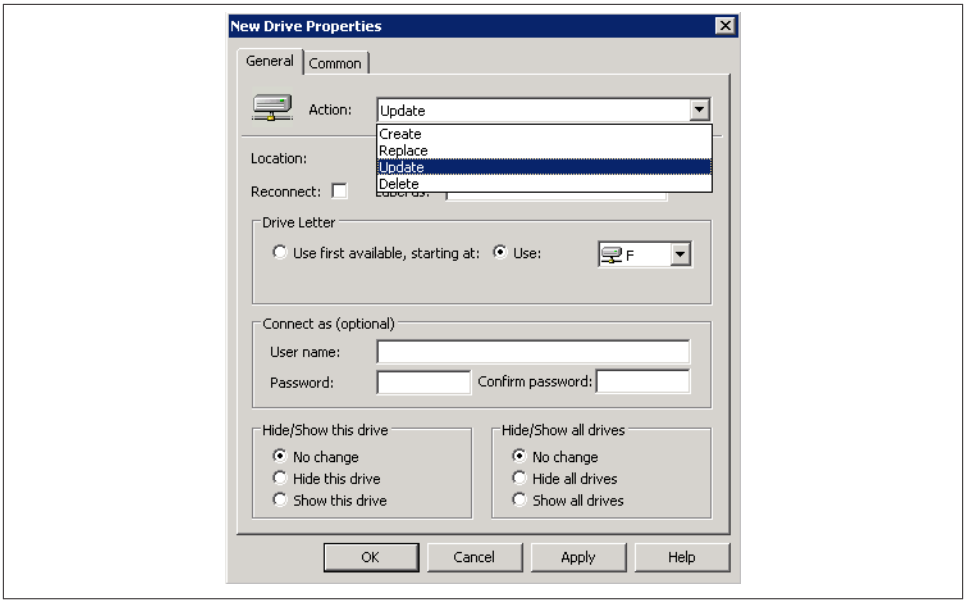


Figure 11-13. Preference action options

You have four options when configuring the behavior of a preference item:

Create
: This will create a new entry.

Replace
: This will remove any related settings on the client machine and overwrite them with the settings configured in the preference item. In the case of a mapped drive, this will un-map a matching mapped drive and remap it.

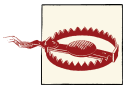
Update
: This will update an existing setting with the settings configured in the preference item, or create the setting if it does not exist on the client machine.

Delete
: This will remove the matching setting from the client machine.

Generally we recommend the use of the Replace option if you want to make sure that the setting on the client machine matches exactly what is configured in the preference item.

Deploying group policy preferences

Before you can start using group policy preferences, you may need to deploy one or more updates to your client workstations. Group policy preferences uses a special component called a client-side extension (CSE) to process the settings. Some versions of Windows include the CSE, and others don't. Table 11-4 shows what versions of Windows will require the installation of the CSE.



If you are currently using the DesktopStandard ProfileMaker product, installing group policy preferences will uninstall ProfileMaker. You will need to migrate your ProfileMaker configuration to group policy preferences as part of the upgrade. Check out *http://bit.ly/YRHXbi* for more information on this process.

Table 11-4. Group policy preferences installation requirements

	Release			
	RTM	**Service Pack 1**	**Service Pack 2**	**Service Pack 3**
Windows XP	Not supported	Not supported	Separate install	Separate install
Windows Server 2003	Not supported	Not supported	Separate install	N/A
Windows Vista	Separate install	Separate install	Included	N/A
Windows Server 2008	Included	Included	Included	N/A
Windows 7	Included	Included	N/A	N/A
Windows Server 2008 R2	Included	Included	N/A	N/A
Windows 8	Included	N/A	N/A	N/A
Windows Server 2012	Included	N/A	N/A	N/A

The CSE is available for download from *http://support.microsoft.com/kb/943729* for platforms that don't include group policy preferences out of the box. The CSE is also offered via Windows Update and the Windows Software Update Service (WSUS) to the relevant versions of Windows.

In addition to the CSE, you will probably want to deploy the latest hotfix rollup for group policy preferences, which fixes a number of bugs and performance issues. As of the time of writing, that hotfix was located at *http://support.microsoft.com/kb/2622802*. Microsoft Product Support Services (PSS) will be able to tell you what the latest version is. If you are using item-level targeting for group membership, you'll want to have *http://support.microsoft.com/kb/2561285* at a minimum to resolve performance issues.

Finally, if you're going to be deploying the group policy preferences CSE to machines running Windows XP SP2 or Windows Server 2003 SP1 or SP2, you will also need to deploy XMLLite to those machines. You can get XMLLite from *http://support.microsoft.com/kb/915865*.

Item-Level Targeting

One of the really powerful features of group policy preferences is known as *item-level targeting* (ILT). ILT allows you to control the application of settings on a per-item basis based on granular expressions that rely on dozens of properties of the user and computer to which the policy applies. Normally, to filter what settings apply to different sets of users or computers, you need to create separate GPOs and then apply security filtering or a WMI filter. With ILT, you can filter individual preferences within a GPO.

To configure ILT for a preference, open Properties, then select the Common tab. On the Common tab, check "Item-level targeting" as shown in Figure 11-14, and click Targeting.

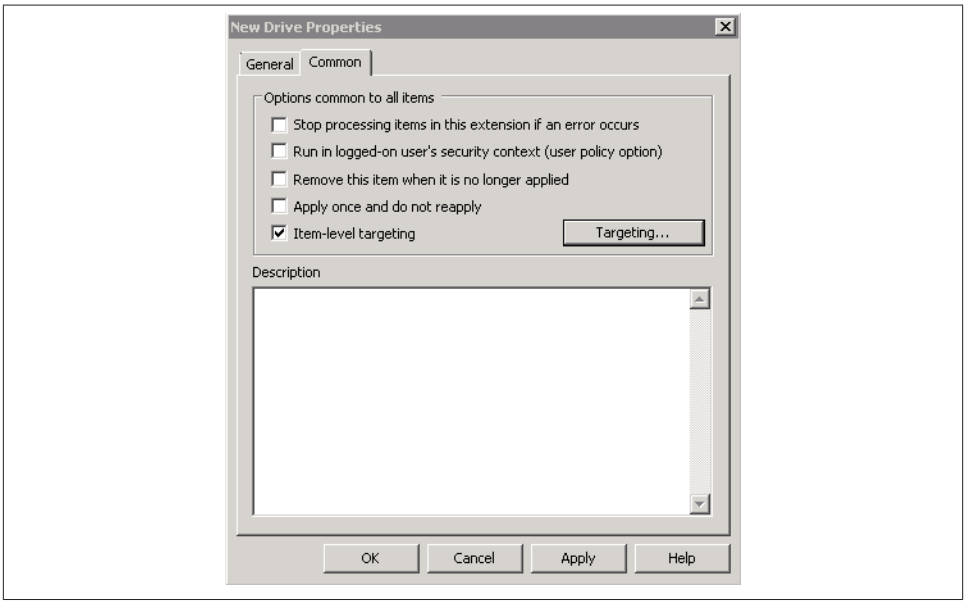


Figure 11-14. Mapped drive preference settings

Figure 11-15 shows a sample usage of ILT. In this example, users who are in the Executives group, and users who are both in the Finance Department and are Headquarters Employees will receive the preference. Other users will not.

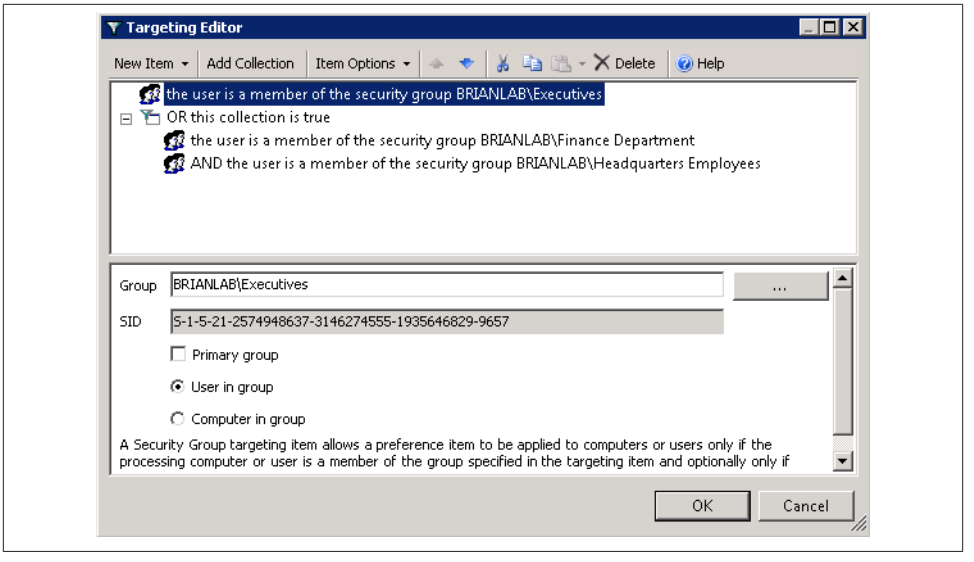


Figure 11-15. Sample item-level targeting

Running Scripts with Group Policy

Logon scripts have been a facet of Windows domains since Windows NT. You can still define a single logon script to be run on the properties of a user account in Active Directory, and Windows clients will run the script at logon; however, Group Policy offers a much richer experience for logon scripts. You can run multiple scripts at logon, and in addition you can define scripts that run during logoff, computer startup, and computer shutdown. While this functionality is generically referred to as "scripts," you can run batch files, Visual Basic (VB) scripts, executables, or Windows PowerShell scripts.

In order to run a PowerShell script at user logon/logoff or computer startup/shutdown, the client machine must be running Windows 7/Windows Server 2008 R2 or newer.

You can specify one or more logon or logoff scripts in a Group Policy with the GPME. The logon and logoff script settings are located under *User Configuration\Policies\Windows Settings\Scripts (Logon/Logoff)*. Double-click the Logon node to configure logon scripts. On the Scripts node you can specify any type of script or executable except Windows PowerShell scripts. PowerShell scripts are specified on the PowerShell Scripts node. The only difference in configuring PowerShell scripts versus standard scripts is that Group Policy allows you to define whether PowerShell scripts are run before or after the scripts/items specified on the Scripts tab.

You can find startup and shutdown scripts inside the GPME by browsing to *Computer Configuration\Policies\Windows Settings\Scripts (Startup/Shutdown)*.

Figure 11-16 shows a single VB script configured to run at logon. When you configure any type of script (logon/logoff or startup/shutdown), you will need to store the file (and any dependencies) in a location that is accessible to all of the users or computers that will be executing the script. Generally speaking, the easiest place to store these files is inside the Group Policy template (GPT) folder in Sysvol. By placing the files inside Sysvol, you ensure that all of the domain controllers in the domain will be able to replicate the files alongside the GPOs they are associated with.

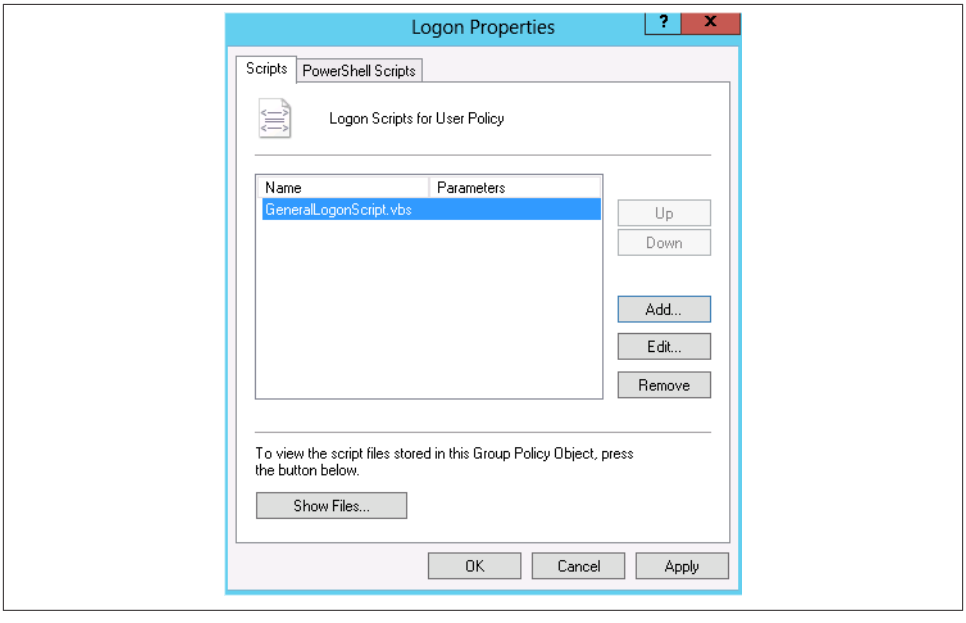


Figure 11-16. Logon script configuration

If you click the Show Files button in Figure 11-16, Windows Explorer will open to the scripts folder specific to the GPO you are editing and the type of script. You can use this handy shortcut to drag and drop files into the GPT location in Sysvol.

Logon (and startup) scripts are extremely common in organizations that have adopted Group Policy. These scripts are often used to configure common user experience settings such as drive mappings or printers, or perhaps to update the configuration of computers in the domain. The downside of running scripts at logon or computer startup is that they can cause noticeable performance issues.

Even the best-written scripts can encounter a scenario that the script writer didn't think of, and when that scenario comes about, computer performance can be adversely affected. For this reason, we have placed the section on scripts and Group Policy after the section on group policy preferences. In our experience, the vast majority of tasks administrators are performing with a logon or startup script can be completed with group policy preferences. With this in mind, wherever possible, we recommend the use of group policy preferences over logon scripts and startup scripts.

Group Policy Modeling

While the Group Policy Results Wizard allows you to see the effective settings for a user and/or computer, it is dependent on that user having logged into the computer in question before. The GPMC includes a second wizard that is helpful for generating "what-if" reports for various scenarios. You can specify a specific user and computer to model or specific locations in the OU hierarchy, as shown in Figure 11-17.

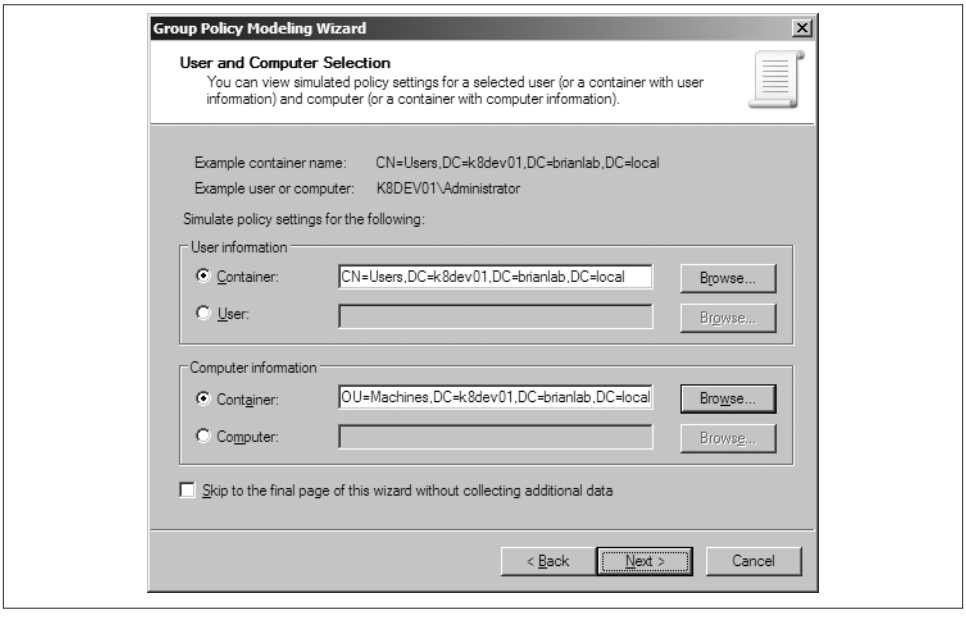


Figure 11-17. Group Policy Modeling Wizard

The wizard will allow you to simulate conditions such as slow links, loopback processing, or site affinity, as shown in Figure 11-18. You can also simulate a change to the user or computer's security group membership if, for example, you are testing changes to security filtering for a policy. Figure 11-19 shows the report that is created by the GPMC. The Summary and Settings tabs are similar to the results reports, and the Query tab displays a summary of the parameters you specified in the modeling wizard.

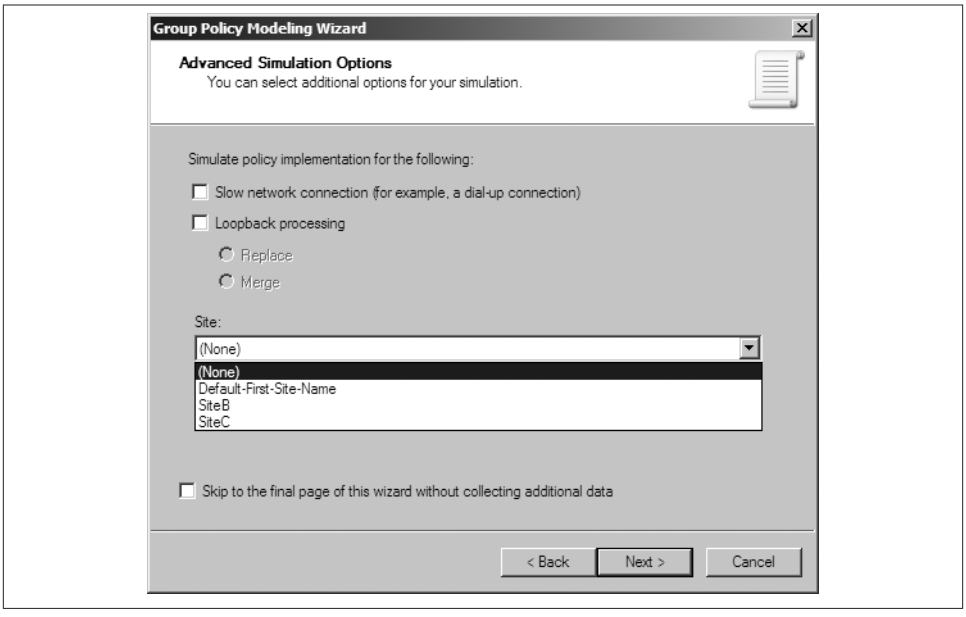


Figure 11-18. Simulating network conditions and loopback processing

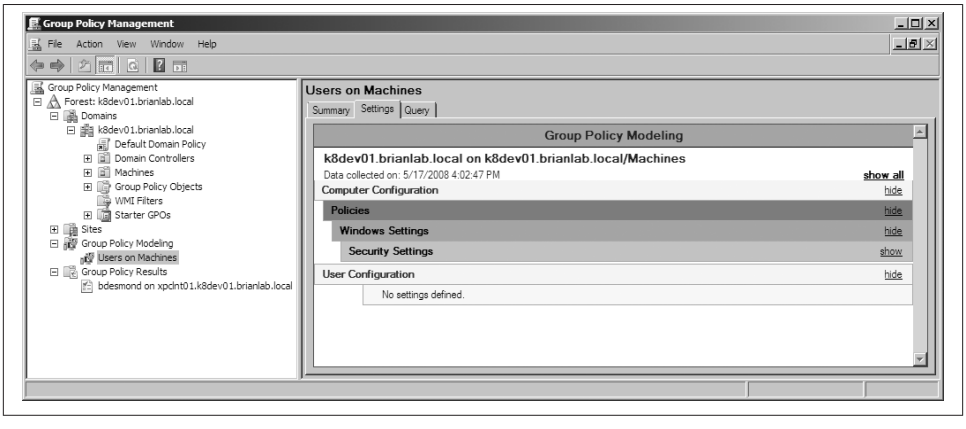


Figure 11-19. The Group Policy modeling report

Like with the Group Policy results reports, you can rerun and export modeling reports as well as view the RSoP MMC by using the "Advanced View" right-click option.

Delegation and Change Control

When planning your group policy deployment, you have to work out how you will maintain firm control over GPOs once you start deploying them. Specifically, you need to consider who will be managing your GPOs and how you will keep control over the wide-ranging changes they can make.

The importance of change-control procedures

The best way to keep track of GPOs in your organization is through a series of change-control procedures. These work well whether your GPO administrators are domain administrators or not. The complexity of the change-control system used for group policies varies from organization to organization. Some organizations opt to manage all of their group policies in their configuration management database (CMDB); others implement simpler systems such as Word documents or Excel spreadsheets. It is important that you test any group policy changes in the lab before applying them in production, due to the potential for disrupting service if a faulty policy change is applied. Microsoft offers a tool called Advanced Group Policy Management (AGPM, part of the Microsoft Desktop Optimization Pack [MDOP]) to assist with implementing group policy management permissions and change-control procedures.

Designing the delegation of GPO administration

There are three types of permission that can be considered here:

- The permission to allow sets of users to link policies (and manage the block inheritance setting) to a domain or an organizational unit branch
- The permission to allow sets of users to create GPOs
- The permission to allow sets of users to change the GPOs themselves

Link delegation can be easily accomplished using the Delegation of Control Wizard that you can access by right-clicking an organizational unit, domain, or site in the Active Directory MMC and choosing Delegate Control. You'll want to use the "Manage Group Policy links" task. Here you are actually delegating read and write access to the `gPLink` and `gPOptions` attributes of objects. If you're interested in how these attributes work, set up a few GPOs in your Active Directory environment and use ADSI Edit to examine the attributes of the OUs to which you link the GPOs.

We discuss the Delegation Control Wizard in detail in Chapter 16.

Permission to create GPOs is limited to those accounts indicated in the upcoming sidebar "Default GPO Permissions" on page 324 by default. However, you can add users to the Group Policy Creator Owners security group, and members of that group can create new GPOs.

For information on modifying the default GPO permissions, reference Microsoft Knowledge Base article 321476 at *http://support.microsoft.com/kb/321476*.

If a member of Group Policy Creator Owners creates a GPO, that user is set as the Creator Owner[2] of the GPO and can continue to manage it. The Creator Owner of a GPO can manage the GPO even if that user is removed from all groups that give GPO management privileges.

GPC data in Active Directory (i.e., the actual Active Directory object itself) is configured not to inherit security permissions from its parent containers.

You can delegate edit access to the creators of new GPOs by placing those users into the Group Policy Creator Owners group. If you also want to delegate edit access to more people or to GPOs that a user is not the Creator Owner of, use the GPMC:

1. Navigate to the *Group Policy Objects* folder under the domain in which the GPO you want to grant permission to is contained.
2. Click on the GPO you want to delegate and open the Delegation tab in the right pane, as shown in Figure 11-20.
3. Click the Add button to bring up the object picker.
4. Select the users or groups you want to have access.
5. Pick the permissions you want to grant. You have three options:
   - Read
   - Edit settings
   - Edit settings, delete, modify security
6. Finally, click OK and the delegation will be applied.

2. When administrators create GPOs, the Domain Admins group becomes the Creator Owner.

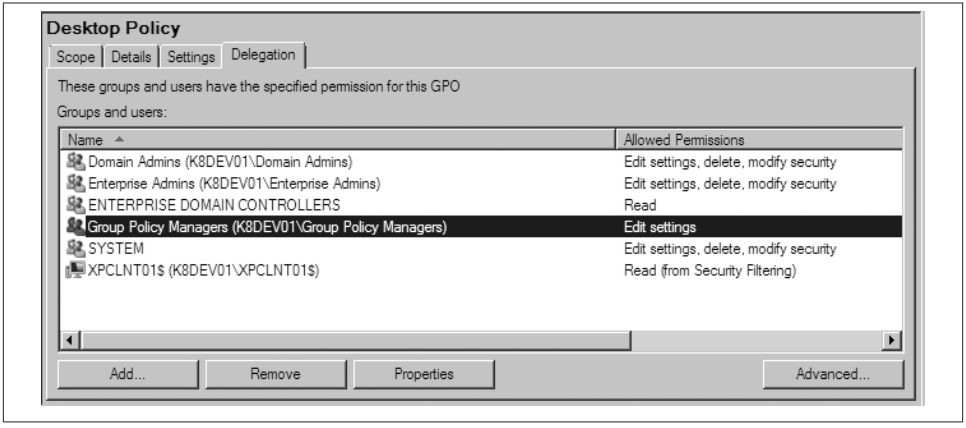
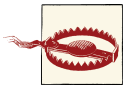


Figure 11-20. GPO delegation in the GPMC



A word of warning before we finish up here: correctly applied, GPOs are fundamental to the well-being of your Active Directory environment and your network at large. Policies incorrectly applied to the root of the domain could lock down the Administrator account or disallow logons at domain controllers. This is obviously a worst-case scenario, but it has happened. More common are mistakes such as a mistyped registry value that forces users to an invalid proxy server and thus stops Internet Explorer from working, or forgetting to add a security filter and accidentally applying a policy to all machines (the default) when it was only meant to apply to a few machines.

These changes have the potential to affect all users or computers throughout the tree, so we would caution you to keep GPO administrators to a very select subset. If you allow non-administrators the ability to create, change, and delete GPOs, they have to be able to take responsibility for and be accountable for their actions. Users who are already administrator-equivalent will automatically be able to administer GPOs and should already be held accountable.

Default GPO Permissions

Any user, computer, or group needs both Read and Apply Group Policy permissions to apply a policy. Active Directory ships with certain permissions already in existence for GPOs. These are:

- The Authenticated Users group has Read and Apply Group Policy.
- The Creator Owner has Full Control without an explicit Apply Group Policy.

- The Local System group has Full Control without an explicit Apply Group Policy.
- The Domain Admins group has Full Control without an explicit Apply Group Policy.
- The Enterprise Admins group has Full Control without an explicit Apply Group Policy.
- The Group Policy Creator Owners group has Full Control without an explicit Apply Group Policy.

Administrators in the latter three groups are also Authenticated Users, and as a result they inherit the Read permission from that group. If you don't want administrators to have the user parts of GPOs applied on logon, set the Apply Group Policy permission to Deny for Domain Admins, Enterprise Admins, and possibly Creator Owner as well.

Using Starter GPOs

One of the new features for Group Policy beginning with the Windows Server 2008 toolset is *Starter GPOs*. Starter GPOs allow you to define a template GPO that can then be duplicated in the GPMC. Starter GPOs are stored and replicated alongside normal group policies in the Sysvol container, in a new folder called *StarterGPOs*.

The first time you access the Starter GPOs node in the GPMC, you will be prompted to create the *StarterGPOs* folder in Sysvol. Starting with Windows Server 2008 R2, a series of read-only Starter GPOs are automatically created. These Starter GPOs include the settings necessary to implement the Windows XP SP2 and Windows Vista Security Guides from Microsoft. At the time of writing, there were no comparable Starter GPOs for Windows 7 or Windows 8. In addition, read-only Starter GPOs to enable the necessary ports for Group Policy remote management through the Windows Firewall are included.

The settings you can define in custom Starter GPOs are limited to the settings in the Administrative Templates section of a GPO.

Creating a Starter GPO is nearly identical to creating a normal GPO. Simply right-click the Starter GPOs virtual container in the GPMC and click New.

You may have noticed that the New GPO dialog has an additional selection for the Starter GPO, as shown in Figure 11-21. If you select a Starter GPO at this time, the settings from that Starter GPO will be imported into your new GPO, providing a baseline to start with.

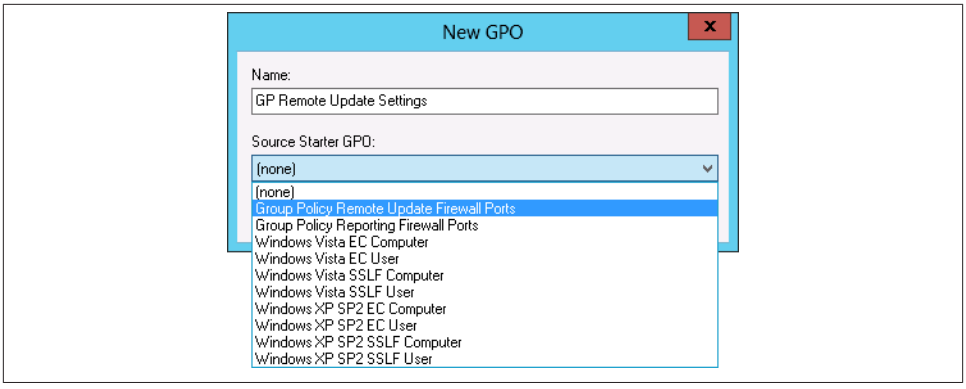


Figure 11-21. New GPO with Starter GPO

Group Policy Backup and Restore

Restoring the Sysvol container from a system-state backup of a domain controller is an involved and somewhat painful process. Fortunately, with the introduction of the GPMC, there is an easy GUI and script-based mechanism for backing up and restoring some or all of your group policies. To back up all of the group policies in the domain with the GPMC, right-click on the Group Policy Objects virtual container and select the Back Up All option. You will be presented with a dialog similar to Figure 11-22. You must provide the GPMC with a path to an empty folder to back up all of the GPOs to, and optionally a comment to include in the backup. Once the wizard completes, you can zip the target folder and have a portable and easy-to-work-with backup.

If you want to back up an individual policy, simply right-click it under the Group Policy Objects container and select Back Up. The GPMC only supports restoring policies on a per-policy basis. To restore a policy, right-click it under the Group Policy Objects container and select "Restore from Backup." The wizard will prompt you for the path to the backup. You must point the wizard to the *parent* folder of the backup. Thus, if you were restoring from the backup created in Figure 11-22, you would point the restore wizard to *C:\GPOBackup*.

If you need to restore all of the GPOs in a backup at once, you can use the *RestoreAllGPOs.wsf* script that is included in the GPMC sample scripts discussed in the "Scripting Group Policy" on page 327 section of this chapter. Similarly, you can use the *BackupAllGPOs.wsf* script to back up all of the GPOs in a domain. There are also individual group policy backup and restore scripts called *BackupGPO.wsf* and *RestoreGPO.wsf*, respectively. In addition to these scripts, you can take advantage of the *Backup-GPO* and *Restore-GPO* cmdlets included with Windows Server 2008 R2 and newer.

You can also use the GPMC backup and restore functionality to copy group polices between domains, or perhaps between a test environment and production. The GPMC

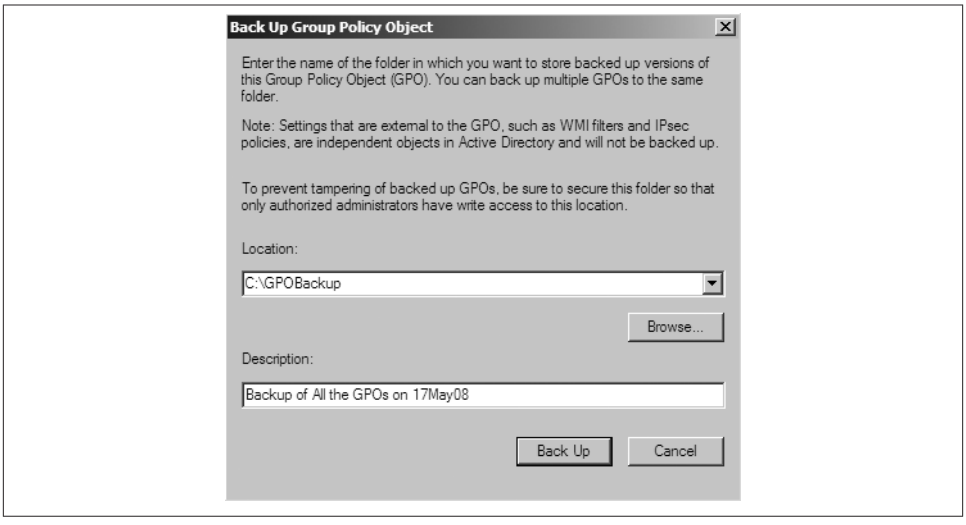


Figure 11-22. Group Policy backup

includes a Migration Table Editor tool that will be invoked when you need to modify references to security principals in the source domain so that the policy will continue to function in the target domain. For more information on migrating group policies, see the whitepaper Migrating GPOs Across Domains with GPMC.

Scripting Group Policy

Historically, one of the hurdles to efficiently managing GPOs with Active Directory has been the lack of scripting support. The GPMC provides a number of scripting capabilities via COM-based objects that can be accessed from VB scripts, PowerShell, and so forth. Windows Server 2008 R2 introduced a number of PowerShell cmdlets for use with Group Policy, and Windows Server 2012 expanded on this selection.

There are a number of useful sample scripts (over 30) for working with the GPMC that Microsoft distributes freely on its website. Once you download and run the installer, the scripts will be stored in *%ProgramFiles%\Microsoft Group Policy\GPMC Sample Scripts*.

The following is a partial list of some of the tasks you can perform via scripts with the GPMC objects:

- Create a GPO with the default settings.
- Copy a GPO.
- Import GPO settings.
- Set GPO permissions.
- Delete a GPO.
- Search for GPOs.
- List GPOs.
- Retrieve GPO information.
- Back up GPOs.
- Restore GPOs.
- Generate a RSoP report for GPOs.

The PowerShell cmdlets that come with Windows Server 2008 R2 and Windows Server 2012 enable you to script the creation and management of GPOs as well as report on and script settings that are set via administrative templates. You cannot script settings that are set with client-side extensions, such as for the Windows Firewall or Restricted Groups. Table 11-5 provides a listing of the PowerShell cmdlets for Group Policy as well as their purposes.

Table 11-5. Group Policy PowerShell cmdlets

PowerShell cmdlet	Purpose
Backup-GPO	Export a GPO or all the GPOs in the domain to the local filesystem.
Block-GPInheritance	Configure an OU not to inherit Group Policy settings.
Copy-GPO	Duplicate a GPO with a new name. This cmdlet can also be used to copy GPOs across domains.
Get-GPInheritance	Display the Group Policy inheritance settings for an OU.
Get-GPO	Display information about a group policy object.
Get-GPOReport	Generate a report in HTML or XML format detailing the contents of a GPO.
Get-GPPermission	List a permission entry in the ACL on a GPO.
Get-GPPermissions	List the permissions specified on a GPO (e.g., for security filtering or delegation of editing).
Get-GPPrefRegistryValue	Display group policy preferences registry settings specified in a GPO.
Get-GPRegistryValue	Display Group Policy settings that are registry-based. Effectively, this is all of the settings in the Administrative Templates section of a GPO.
Get-GPResultantSetOfPolicy	Run RSoP and display the results for a user and/or computer.
Get-GPStarterGPO	Display information about a Starter GPO.
Import-GPO	Import one or more exported GPOs from the filesystem into an existing GPO. These exported GPOs would be created with the *Backup-GPO* cmdlet or the associated Back Up option in the GPMC.
Invoke-GPUpdate	Refreshe group policies on the local machine or remotely on a target machine.

PowerShell cmdlet	Purpose
New-GPLink	Link a group policy to an OU, domain, or site.
New-GPO	Create a new GPO.
New-GPStarterGPO	Create a new Starter GPO.
Remove-GPLink	Remove a group policy link from an OU, domain, or site.
Remove-GPO	Delete a GPO from Active Directory.
Remove-GPPrefRegistryValue	Remove one or more group policy preferences registry settings from a GPO.
Remove-GPRegistryValue	Remove one or more registry settings from a Group Policy. Effectively, this is any of the settings in the Administrative Templates section of a GPO.
Rename-GPO	Rename a GPO in Active Directory.
Restore-GPO	Restore one or more GPOs that were backed up with *Backup-GPO*.
Set-GPInheritance	Toggle the Group Policy inheritance settings for an OU.
Set-GPLink	Modify the link order (precedence), "enforced" flag, or link status (enabled/disabled) for a group policy link on an OU, domain, or site.
Set-GPPermission	Modify the permissions on a GPO (e.g., for security filtering or delegation of editing).
Set-GPPermissions	Modify the permissions specified on a GPO (e.g., for security filtering or delegation of editing).
Set-GPPrefRegistryValue	Modify one or more group policy preferences registry settings in a GPO.
Set-GPRegistryValue	Modify one or more registry settings in a Group Policy. Effectively, this is any of the settings in the Administrative Templates section of a GPO.

In addition to the cmdlets included with Windows, there are a number of third-party management extensions for Group Policy that deliver a great deal of value. One company that offers a number of PowerShell-based solutions is SDM Software at *http://www.sdmsoftware.com*.

Troubleshooting Group Policy

If at any point you need to debug group policies, there are a few options available. We will look at a few of them here, including the Resultant Set of Policy (RSoP) tool. We'll also see how to force a group policy update and explore the various logging options that are available, depending on what version of Windows the client machine is running.

Group Policy Infrastructure Status

The Windows Server 2012 GPMC includes a new health report for Group Policy that focuses on Active Directory and Sysvol replication status. The health report checks the status of each group policy in the domain and ensures that the policy and its associated metadata are consistent across all of the domain controllers in the domain. In order to do this, the GPMC selects one domain controller as the baseline and then compares each domain controller against that baseline.

You can access the infrastructure status by clicking on a domain in the GPMC and then opening the Status tab. Figure 11-23 shows the status for a domain that is in good health.

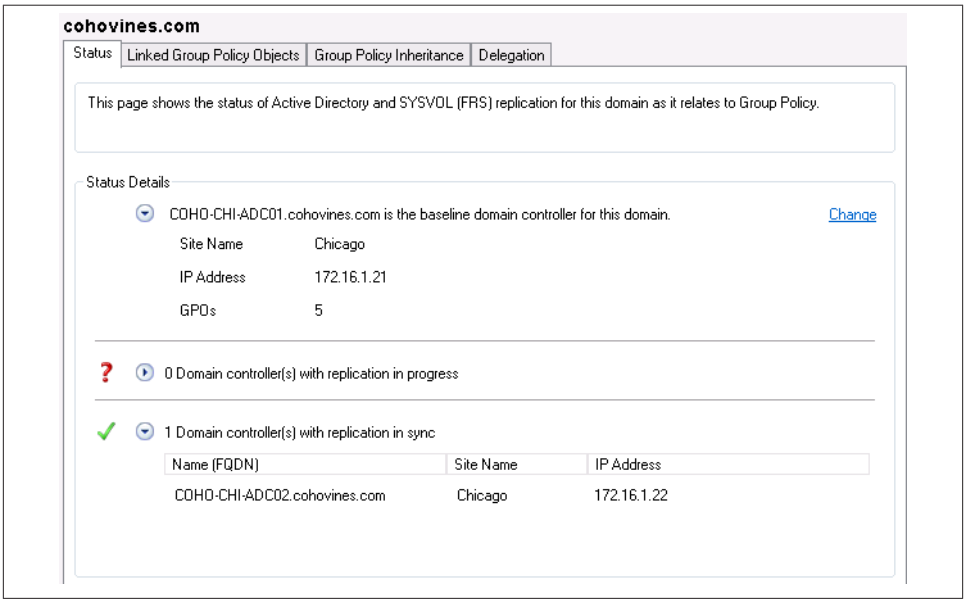


Figure 11-23. Group policy infrastructure status

Group Policy Results Wizard

The Group Policy Results Wizard is a frontend for the Resultant Set of Policy tool. RSoP is a very powerful tool that will identify what GPO settings have been 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.

Windows Server 2012 improves the RSoP reporting to include a number of additional data points. These include the processing time for each CSE in the policy, the policy that contributed each setting, loopback processing and link speed data, and whether inheritance blocking was a factor in the resultant policy set.

To start the Group Policy Results Wizard, open the GPMC, right-click the *Group Policy Results* folder, and launch the wizard. Figure 11-24 shows the initial screen, where you will be prompted to select a target machine to run the RSoP session on.

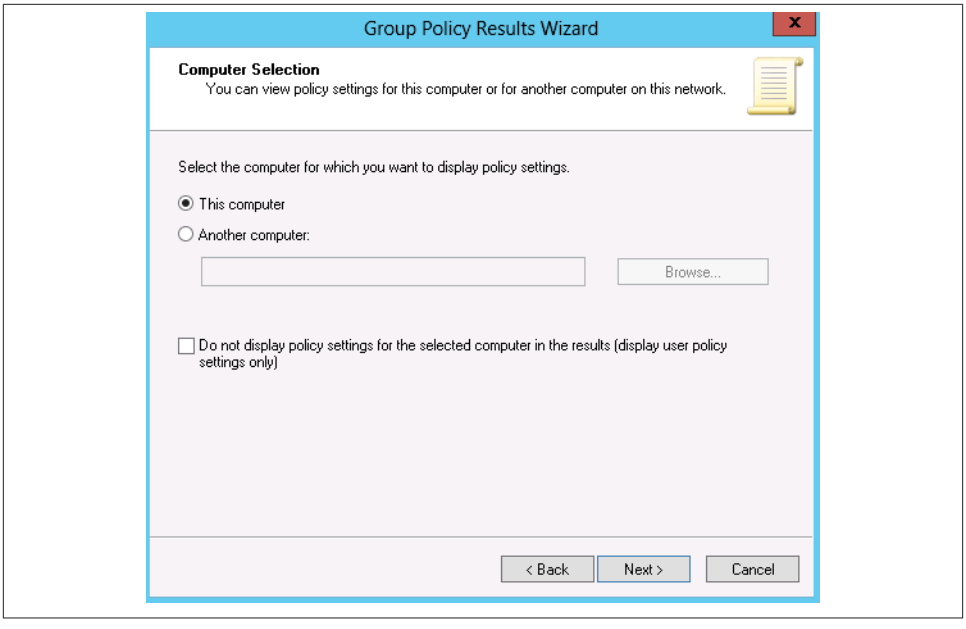


Figure 11-24. Group Policy Results Wizard—target computer selection

The next screen, as shown in Figure 11-25, allows you to optionally specify a particular user for whom to display the policy settings. If you only need computer settings, you can skip this step. The wizard will only display users who have logged onto the target machine successfully in the past.

Once you finish the wizard, the GPMC will create a report similar to the one shown in Figure 11-26. This report is divided into three sections:

- The Summary tab gives a high-level overview of what factors affected policy application, filters that were applied, and so forth.
- The Details tab allows you to view every setting that was applied and the GPOs from which they originated.
- The Policy Events tab gives you a consolidated view of event log entries related to group policy processing.

If you are looking for the Resultant Set of Policy view rather than the HTML report style pictured in Figure 11-26, right-click the report and click Advanced View.

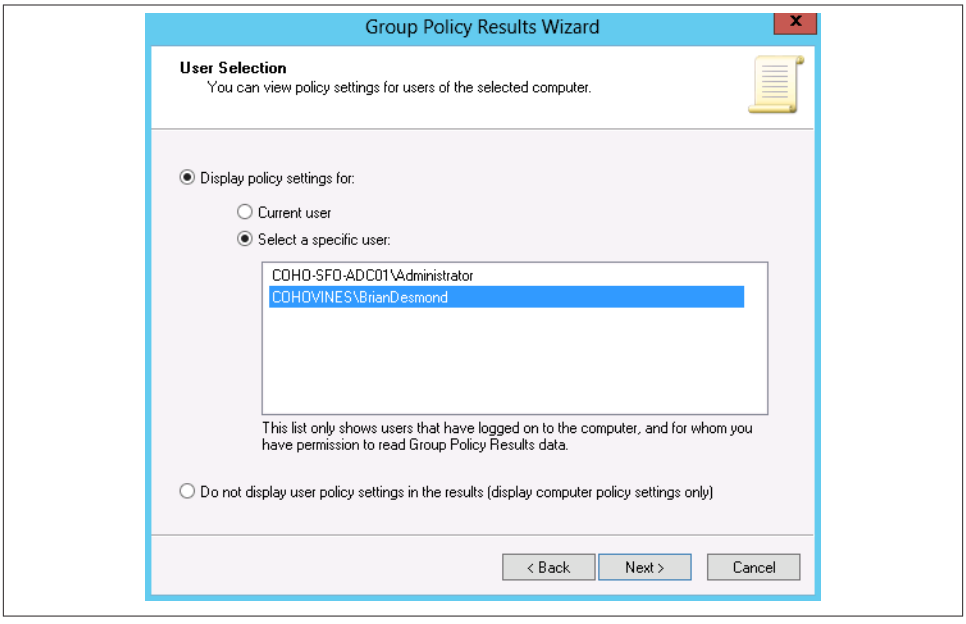


Figure 11-25. Group Policy Results Wizard—target user selection

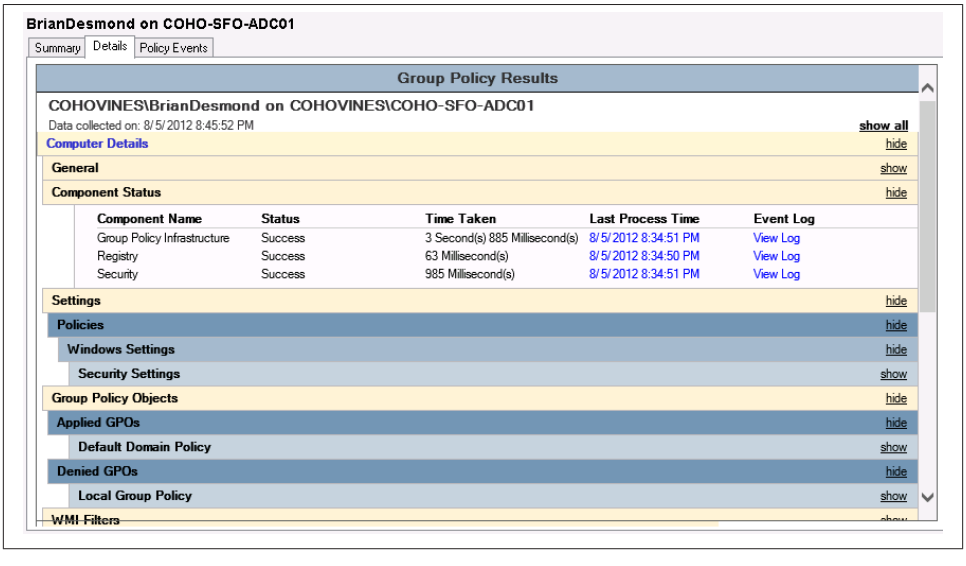


Figure 11-26. Group Policy Results report

One of the nice features of the Group Policy Results tool is that the reports are saved, so you can reference them later. You can also right-click the report in the tree at the left and rerun the query or export the report.

You can also run an RSoP session from the command line; see the sidebar "Using the GPResult Tool" on page 333.

Using the GPResult Tool

The *gpresult* tool can be run from any system to export the group policy results for the current session or a target machine and/or user to a text file. *gpresult* has been available since Windows XP.

To export the group policy settings for the current session, use this command:

```
gpresult /v
```

If you would like to target a different user, run:

```
gpresult /v /USER otherusername
```

To target a different machine, run:

```
gpresult /v /S othercomputer
```

You can also combine these switches to target a different user and computer.

Starting with Windows Vista/Windows Server 2008, you can use the */H* switch to export an HTML report similar to the one found in the GPMC console. The */X* switch will export a report in XML format. You can use the XML-formatted report to automate processing of the results with a script.

For additional options, run `gpresult /?`.

Forcing Group Policy Updates

Sometimes when troubleshooting an issue, you may not want to wait for a policy to refresh in the normal time period. Fortunately, there is a command-line tool to force an update called *gpupdate*. You can simply run `gpupdate` to update the group policy settings; however, there are other options available that are documented if you run `gpupdate /?`. Windows 8 and Windows Server 2012 clients also include the *Invoke-GPUpdate* PowerShell cmdlet that offers the same functionality as *gpupdate*.

The version of GPMC included with Windows Server 2012 (and the accompanying PowerShell cmdlets) introduces the ability to remotely refresh group policies on one or more machines. This feature works by remotely creating a scheduled task on the target machine(s) that refreshes the group policies locally. You can use this feature with clients running Windows Vista or newer and servers running Windows Server 2008 or newer.

In order to schedule the task, the machine you are running GPMC from must have access over the network (e.g., through firewalls) to run remote procedure calls (RPCs) to the target machines. If you are using the Windows Firewall, you will need to either deploy the Group Policy Remote Update Firewall Ports Starter GPO in advance or open the ports and services listed in the Starter GPO on each client.

To remotely refresh group policies for all of the machines in a specific organizational unit, you can right-click the OU in the GPMC and click Group Policy Update. You will receive a confirmation prompt similar to Figure 11-27 that tells you how many machines will be affected by the request. After the request begins, the GPMC will provide you with a live status report, as shown in Figure 11-28. The status report confirms that the group policy refresh has been scheduled on the remote machine but not whether or not the refresh has succeeded. You can export the results to a CSV file for manipulation with a script or Microsoft Excel by clicking the Save button.

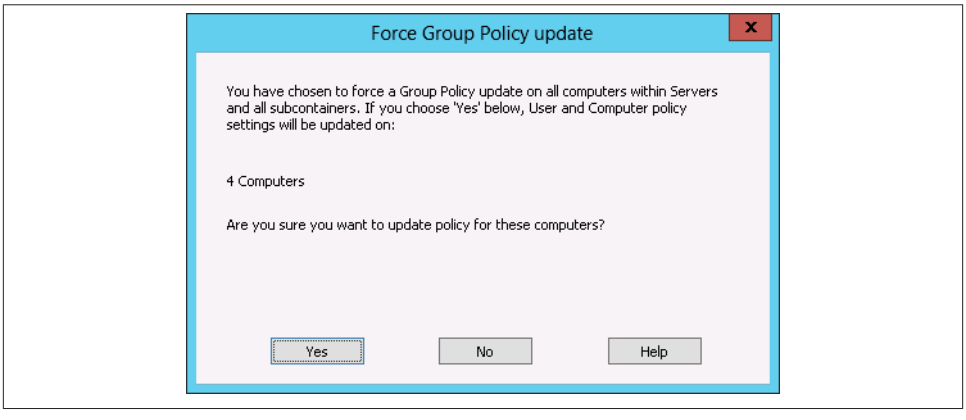


Figure 11-27. Remote group policy update confirmation

Enabling Extra Logging

When Group Policy works, it is an extremely valuable tool to administrators. But as with any technology, from time to time you may need to dig into what's going on behind the scenes in order to troubleshoot an issue with Group Policy. Fortunately, Group Policy comes with a large amount of diagnostic logging that can be enabled when you need to troubleshoot a problem. The format of the logging and how you enable it varies depending on the version of Windows.

Group Policy Logging in Windows 2000, Windows XP, and Windows Server 2003

You can turn on verbose logging in the event log for group policy-related events simply by setting a registry key. Once the key exists with the correct value, logging is done automatically; a reboot is not necessary for the new value to take effect. The value, a

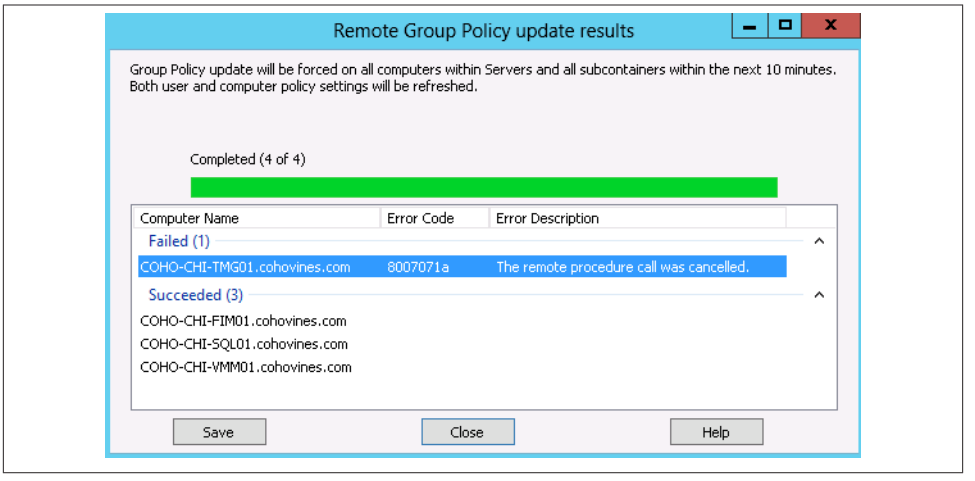


Figure 11-28. Remote group policy refresh status report

`REG_DWORD`, is called `RunDiagnosticLoggingGroupPolicy` and needs to be created with a value of `1` in the `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Diagnostics` key.

The value of `1` sets the logging to verbose mode; setting the value to `0` is the same as having the key absent and is known as normal logging. In other words, the key makes a difference only when set to a value of `1`. It's really as simple as that.

This key is actually one of four currently supported keys that you can use at this location. You also can create the following keys::

- `RunDiagnosticLoggingIntellimirror`
- `RunDiagnosticLoggingAppDeploy`
- `RunDiagnosticLoggingGlobal`

If the verbose logging in the event log is not providing enough information, another option is to enable debug logging for policy and profile processing. To do so, create a value called `UserEnvDebugLevel` as a `REG_DWORD` in the `HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon` key. Assign `UserEnvDebugLevel` the value 10002 in hexadecimal format. Restart the computer, and from then on, extensive logging information will be recorded on the machine in the file *%SystemRoot%\Debug\UserMode\Userenv.log*. For more information, check out Microsoft Knowledge Base article 221833, which can be found at *http://support.microsoft.com/kb/221833*.

The *UserEnv* debug log file is extremely dense, so chances are you will need to search for keywords such as "error" if you are not used to reading the log. Keep in mind that some errors are normal even though they are flagged as errors. Be sure to disable the `UserEnvDebugLevel` setting once you are finished with it.

For a comprehensive listing of group policy logging settings, check out the free ADM file available for download at *http://bit.ly/Xa5Y30*.

Group Policy Logging in Windows Vista/Windows Server 2008 and Newer

Newer versions of Windows publish Group Policy diagnostic information directly to the event log via Event Tracing for Windows (ETW). To view Group Policy tracing information in Windows Vista/Windows Server 2008 or newer, launch the Event Viewer, browse to *Applications and Services Logs\Microsoft\Windows\GroupPolicy*, and open the Operational log. You will find a wealth of information related to the actual processing of group policies as well as performance.

If you're troubleshooting an issue with group policy preferences, you'll need to explicitly enable logging and then parse through the resultant trace files. Microsoft has an excellent article on enabling logging for group policy preferences at *http://bit.ly/Yd8eVH*.

Group Policy Diagnostic Best Practices Analyzer

Microsoft has a free tool available from its website called the Group Policy Diagnostic BPA. BPA stands for *Best Practices Analyzer*, and it's a breed of tool Microsoft has been releasing for various tools and products over the past several years. The Group Policy Diagnostic BPA can help with troubleshooting. To get the most appropriate version of the tool for your environment, go to *http://support.microsoft.com/kb/940122*.

Third-Party Troubleshooting Tools

Independent software vendor SDM Software has developed a Group Policy troubleshooting toolset that can visually report on Group Policy processing health on clients, and visually analyze many of the logging options discussed earlier in this chapter. For more information, visit *http://www.sdmsoftware.com/products/freeware/*.

The GPO Guy community website is also an excellent resource for Group Policy information. The site provides free tools, and hosts an email discussion list for Group Policy topics.

Summary

One of the big selling points of Active Directory has always been Group Policy. In this chapter, we covered the details of how group policies are stored in Active Directory, how GPOs are processed by clients, the GPO precedence order, the effects of inheritance, and the role ACLs play.

There are a number of tools available for managing Group Policy. 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 necessary from a single interface. Another benefit of the GPMC is that it installs several COM objects that allow you to script many of your GPO management functions.

There are a number of troubleshooting and diagnostic tools available for discerning what happened when things go wrong that we introduced in this chapter as well.

CHAPTER 12

Fine-Grained Password Policies

Undoubtedly, one of the most exciting new features in Windows Server 2008 Active Directory was the introduction of a feature called *fine-grained password policies* (FGPPs). Prior to FGPPs, domain account policies (password and lockout policies, specifically) could only be set on a per-domain basis. If you had a requirement to have separate password-complexity requirements for different sets of users, you could either deploy a third-party password filter or deploy additional domains. Fine-grained password policies solve both of these issues within a single domain and are immediately available once your domain is running at the Windows Server 2008 or better domain functional level.

Understanding Password Settings Objects

Fine-grained password policies you create are represented by *password settings objects* (PSOs) within Active Directory. PSOs are standard Active Directory objects and are stored under the System container in the domain partition.

Fine-grained password policy functionality is available beginning with Windows Server 2008, and as such, Windows Server 2003 and earlier versions of Windows domain controllers are not capable of enforcing this functionality. FGPPs become available once the domain is running at the Windows Server 2008 or better domain functional level. While you can create and manage PSOs before your domain is running at this functional level, the policies will have no effect on users.

The easiest ways to manage PSOs are with the Windows Server 8 Active Directory Administrative Center (ADAC), with Windows PowerShell, or with a command-line tool. We'll cover management of PSOs with ADAC and a free command-line tool called *PSOMgr*, available at *http://www.joeware.net/freetools/tools/PSOMgr/*.

If you're not familiar with ADAC, take a look at Chapter 3.

Scenarios for Fine-Grained Password Policies

Nearly all of the settings that you could historically only define in the Default Domain Policy for the entire domain can be configured with password settings objects. The exceptions to this rule are the Kerberos settings that must still be defined on a per-domain basis via Group Policy.

If you are using one or more custom password filters in your domain, you can continue to utilize these filters in addition to fine-grained password policies.

Each of the settings in question is stored as an attribute of the `msds-PasswordSettings` schema class that was added with the Windows Server 2008 schema extensions. These attributes are outlined in Table 12-1.

Table 12-1. Mandatory password settings object attributes

Attribute	Description
`cn`	The name of the PSO
`msDS-PasswordSettingsPrecedence`	The order of precedence of the PSO in the event that multiple PSOs apply to a user
`msDS-PasswordReversibleEncryptionEnabled`	Toggles storing the password with reversible encryption
`msDS-PasswordHistoryLength`	The number of previous passwords stored in Active Directory
`msDS-PasswordComplexityEnabled`	Toggles password complexity checking
`msDS-MinimumPasswordLength`	The minimum length of the password
`msDS-MinimumPasswordAge`	The minimum interval before the password can be reset
`msDS-MaximumPasswordAge`	The maximum age of the password before it must be reset
`msDS-LockoutThreshold`	The number of failed login attempts necessary to trigger a lockout
`msDS-LockoutDuration`	The number of minutes to lock the account out
`msDS-LockoutObservationWindow`	The time window during which the lockout threshold is maintained

Defining Password Settings Objects

When you define your strategy for using password settings objects, you will need to determine the number of separate PSOs you will require to implement your strategy, as

well as the relevant values for each PSO. All of the settings for a given user will come from a single PSO, as the settings from multiple PSOs are not merged. This may mean that you will need to duplicate some common settings across multiple PSOs.

Generally speaking, you should endeavor to have as few PSOs defined as possible in order to implement your password policies. PSOs add management overhead, and thus each additional PSO will add complexity to your environment. We will discuss the delegation options for PSOs in depth later in this chapter, but keep in mind that, generally speaking, the team that runs the domain will probably be responsible for managing the PSOs; however, the helpdesk staff also need to be aware of them so that they can respond correctly to questions from users, such as "What are the rules for the password that I should enter?" Some organizations may elect to delegate this responsibility to an information security group instead.

As you begin to build your plan for deploying fine-grained password policies, keep in mind that as a general best practice you should aim to limit the number of PSOs in a domain to the minimum number you really need.

Defining PSO precedence

Password settings objects are associated with users either directly or via group membership. The `msDS-PSOAppliesTo` attribute of each PSO is a forward link that can be linked to either users or global groups. Since the `msDS-PSOAppliesTo` attribute is multivalued, a single PSO can be applied to multiple users or groups simultaneously. The `msDS-PSOApplied` attribute is a back-linked attribute of users and groups that defines which PSOs are linked to it. Since `msDS-PSOApplied` is a back link, it is also multivalued, and thus a mechanism must exist to determine which PSO actually applies to a given user.

Each PSO also has an `msDS-PasswordSettingsPrecedence` attribute. This is an integer attribute that must be defined on each PSO and that controls the precedence of the PSO in the event that multiple PSOs apply to a given group. Any integer value greater than or equal to `1` and less than or equal to `2,147,483,646` is valid, and it is recommended that you define a unique precedence value for each PSO in your domain. The lowest precedence value always wins a conflict.

Every user object has an additional constructed attribute called `msDS-ResultantPSO`. When this attribute is returned, it will return the PSO that ultimately applies to the user.

Since `msDS-ResultantPSO` is a constructed attribute, you cannot directly query it in an LDAP filter. If you try to do so, you will receive an error specifying that you used the wrong matching operator.

Domain Controllers use the following logic to compute the resultant PSO when multiple PSOs apply:

1. Is a PSO linked directly to the user? This will override any PSOs linked to the user's groups.
2. Does a PSO apply to any of the user's global security group memberships? If so, the PSO with the lowest precedence wins.

If multiple PSOs with the same precedence are found to apply in step 2, the PSO with the mathematically lowest object GUID wins the conflict.

3. If no PSOs were matched in steps 1 and 2, the Default Domain Policy settings are applied.

Creating Password Settings Objects

Now that we've discussed the basics of password settings objects, we'll go ahead and provision a PSO in the domain in order to walk through the process.

Password settings objects are stored in the Password Settings container under the System container at the root of your domain, as shown in Figure 12-1. You will need to select View→Advanced Features in Active Directory Users and Computers in order to view the contents of the System container.

PSO Quick Start

One useful feature of PSOMgr is its ability to get you started with a copy of your domain password policy (as defined via Group Policy), as well as some common PSO templates. For the sake of simplicity, we'll be working with the common PSO templates PSOMgr creates throughout most of this chapter. In order to create these PSOs, you'll need to run this command: *PSOMgr -quickstart -forreal*. In any case where PSOMgr is going to make changes to your environment, you must specify the *-forreal* switch as a confirmation. If you don't do this, PSOMgr will simply print out the changes it would make if you specified *-forreal*. Figure 12-1 shows the results of running the *-quickstart* option.

Building a PSO from Scratch

While the quick-start examples PSOMgr makes are excellent, chances are you may want to define one or more PSOs with custom settings. In this section, we're going to walk

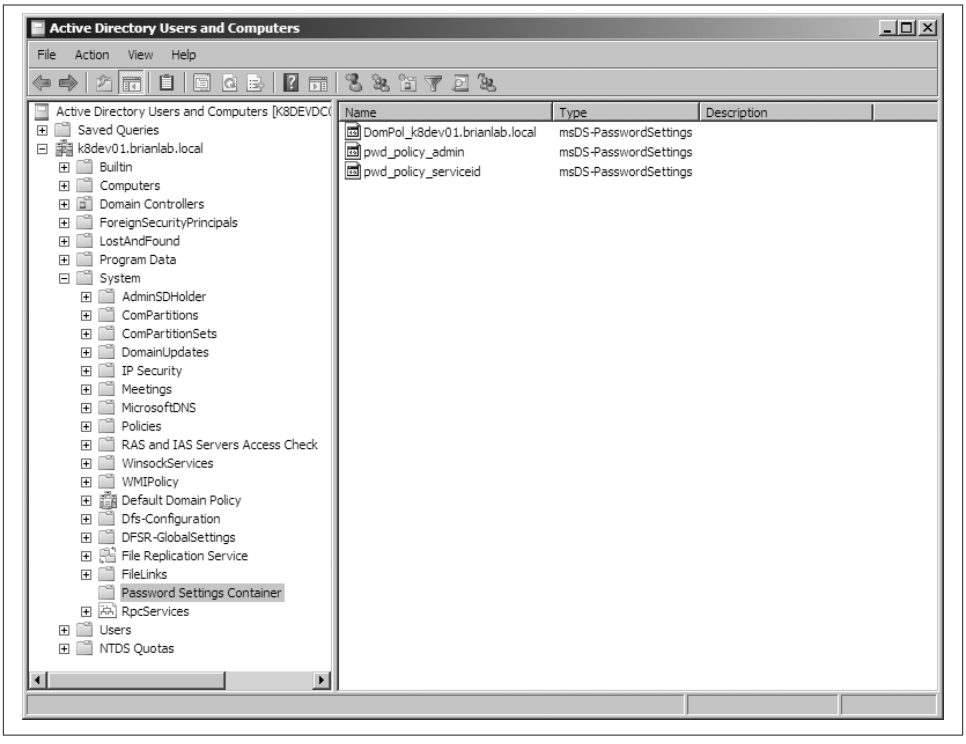


Figure 12-1. Viewing the password settings objects

through creating a PSO with the settings defined in Table 12-2 both with the GUI and with PSOMgr.

Table 12-2. Custom PSO settings

Setting	Value
Name	Presidential Policy
Precedence	11
Reversible Encryption Enabled	No
Password History Length	10 passwords
Password Complexity Enabled	Yes
Minimum Password Length	8 characters
Minimum Password Age	1 day
Maximum Password Age	90 days
Account Lockout Threshold	15 failed logons
Account Lockout Window	30 minutes
Account Lockout Observation Window	30 minutes

Creating a PSO with the Active Directory Administrative Center

In order to create a PSO with the GUI, we'll need to use the Active Directory Administrative Center (ADAC). The ability to create PSOs was added to the Windows Server 8 version of ADAC. Previously, to create PSOs with the GUI, you had to use ADSI Edit.

To begin creating the PSO outlined in Table 12-2, launch ADAC and navigate to *System→\Password Settings Container*. Next, right-click and select New→Password Settings. Complete the dialog as shown in Figure 12-2 and then click OK.

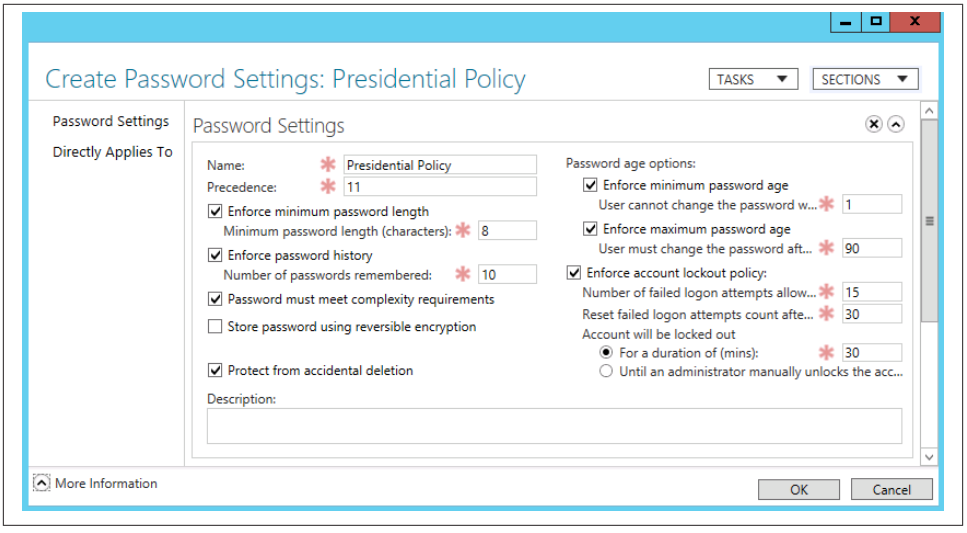


Figure 12-2. Creating a new PSO with the Active Directory Administrative Center

Creating a PSO with PSOMgr

Now that we've created a PSO with ADAC, we'll walk through creating the same PSO specified in Table 12-2 with PSOMgr. There are a number of different ways to specify the same data to PSOMgr. Some of them are easier to read than others, so we'll opt for one of the wordier permutations as opposed to trying to be as succinct as possible.

You can view the full usage screen for PSOMgr by running *PSOMgr -help*, but the switches we'll be using and their arguments are:

- *-add <name>::<precedence>*
- *-lockout <threshold>:<duration>:<observation>*
- *-pwdage <max>:<min>*
- *-pwdlen <minlength>*
- *-pwdhist <historycount>*

- *-pwdcomplex (true|false)*
- *-pwdreverse (true|false)*
- *-forreal*

The command to create the PSO from Table 12-2 is:

```
PSOMgr -add "Presidential Policy"::11 -lockout 15:30:30 -pwdage 90:1 -pwdlen 8
-pwdcomplex true -pwdreverse false -pwdhist 10 -forreal
```

You should, in turn, receive output similar to the following:

```
PSOMgr V01.00.00cpp Joe Richards (joe@joeware.net) April 2007

WARN: Using name Presidential Policy for displayName.

Using host: Chicago\K8DEVDC01.k8dev01.brianlab.local
Retrieving PSOs...
Checking existing policies...

Creating new PSO: CN=CustomPSO1,CN=Password Settings Container,
CN=System,DC=k8dev01,DC=brianlab,DC=local
PSO successfully created.

    Type               : Policy Settings Object
    Domain             : k8dev01.brianlab.local
    Policy Precedence  : 11
    DN                 : CN=Presidential Policy,CN=Password Settings
                         Container,CN=System,DC=k8dev01,DC=brianlab,DC=local
    Name               : CustomPSO1
    Canonical Name     :
    Display Name       : Presidential policy
    Lockout Threshold  : 15
    Lockout Duration   : 30
    Lockout Observation: 30
    Min Pwd Age        : 1
    Max Pwd Age        : 90
    Min Pwd Length     : 8
    Pwd History        : 10
    Pwd Complexity     : TRUE
    Pwd Reversible     : FALSE

The command completed successfully.
```

If you use PSOMgr to create a PSO with a precedence value that is already in use, the operation will succeed but will output a warning to the console similar to the following:

```
WARN: A policy setting object already exists with the same precedence.
WARN: Policy Setting object - DomPol_k8dev01.brianlab.local
WARN: (CN=DomPol_k8dev01.brianlab.local,CN=Password Settings Container,
       CN=System,DC=k8dev01,DC=brianlab,DC=local)
WARN: When a single user has two policies assigned with the same
WARN: precedence at the same scope (i.e. directly applied or
WARN: or applied via group), the policy with the lowest GUID will apply.
```

If you want to edit an existing PSO with PSOMgr, you can substitute the *-mod* switch in lieu of *-add*. The rest of the switches have the exact same behavior.

Managing Password Settings Objects

The easiest way to manage PSOs is with ADAC. ADAC exposes all of the options for controlling password and lockout policies as well as for applying PSOs to users and groups. You can also use ADAC to determine what policy applies to a specific user. If you don't have access to ADAC, or you need to script your management tasks, PSOMgr is an excellent command-line alternative that we'll also explore.

Strategies for Controlling PSO Application

There are a few strategies for managing which PSO applies to your user population, and you'll have to decide on which strategy works best based on your organization. The first strategy is to define a global security group for each PSO and apply the PSO to that group. You can, in turn, place users (or nest additional global groups) in the group. The second strategy is to apply PSOs directly to users or existing groups, and the third strategy is to mix application to groups and users where necessary.

Applying PSOs to groups

Applying your PSOs primarily to groups is likely to be the most scalable strategy for many organizations. The strategy here is that you create a global group for each PSO and place the users to whom the PSO will apply in this group. While you can reuse existing global security groups, we recommend that you create new groups dedicated just to application of PSOs. This ensures that you will not affect users inadvertently by adding them to or removing them from a group that controls PSOs or resource access.

It is important to remember that only global security groups are applicable for applying a PSO to a user. While you can link a PSO to a universal or domain local group, it will not actually be considered for application to the user in the resultant PSO calculation.

If multiple PSOs apply to a user by virtue of that user being a member of multiple groups linked to different PSOs, the PSO with the lowest precedence will win, and in the case of multiple PSOs with the same precedence, the PSO with the smallest GUID will win.

Applying PSOs to users

If you have a small organization or intend to manage your PSO application with an automated tool, it may be feasible to apply PSOs directly to users. Note that the application of a PSO to a user is controlled by whoever has access to modify the PSO, not the user. The same precedence rules as with groups apply to user-linked PSOs. However, you should never link PSOs directly to users, even with the availability of the precedence facility.

Mixing group application and user application

Taking a hybrid approach to applying PSOs to users and groups may be the recipe for success in your case. Remember that if a user has PSOs applied to him both by groups and directly to his user object, the PSO applied to the user object will always win. With this in mind, you can define exceptions to the rule where the standard PSO for users is implemented with a group, and exception users are linked directly to an exception PSO.

Managing PSO Application

You can manage PSO application with ADAC, ADSI Edit, Active Directory Users and Computers (ADUC), or PSOMgr. You control PSO application by editing the `msDS-PSOAppliesTo` attribute. We will quickly go over linking a PSO to a group with each of these tools.

Applying a PSO with ADAC

ADAC lets you apply PSOs from a number of locations, including the properties of users, groups, and the PSO itself. We'll go ahead and create a global group called "Presidential Policy PSO Group" to which we'll apply the PSO we created earlier. To apply a PSO to a group, open the properties of the group and then select Password Settings on the left side. Under Directly Associated Password Settings, click Assign and search for *Presidential Policy*. Once you're done, the properties of your group should look similar to Figure 12-3.You can also apply a PSO directly to a user using the exact same steps.

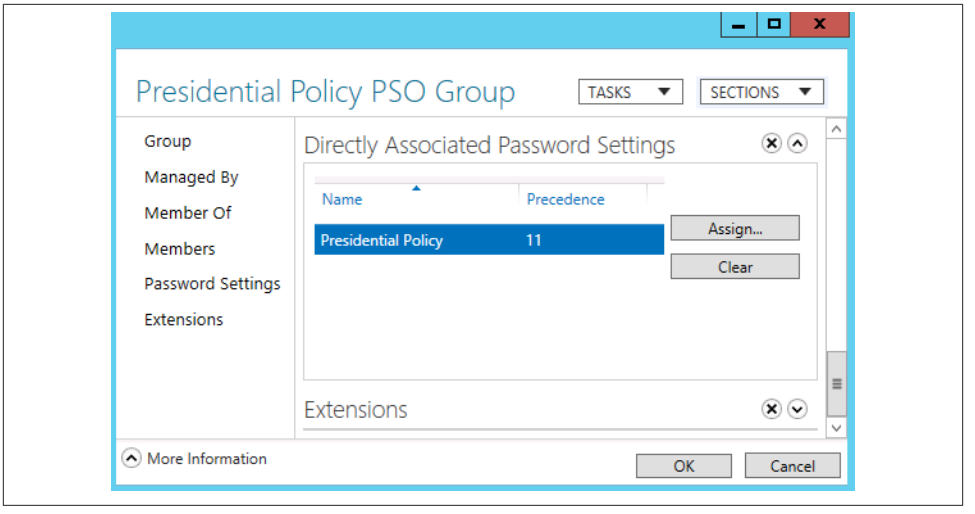


Figure 12-3. Assigning a PSO to a global group

To apply settings from the properties of the PSO (or review all of the users and groups the PSO is directly applied to), you can use the Directly Applies To section and add users or global groups to the list.

Applying a PSO with ADSI Edit

Open the PSO you wish to apply in ADSI Edit by browsing to it under Default Naming Context→DC=domain,DC=com→CN=System→CN=Password Settings Container, or the similar applicable location in your environment. If you do not see `msDS-PSOAppliesTo` in the attribute list, click Filter and uncheck "Show only attributes that have values."

When you edit `msDS-PSOAppliesTo`, a dialog like the one shown in Figure 12-4 appears. This dialog, which was added to ADSI Edit beginning with the Windows Server 2008/Windows Vista RSAT tools, allows you to either manually enter a distinguished name (which was the only option in prior versions) or search for a user or group using the standard Active Directory object search dialog. Use the Add Windows Account button to search Active Directory.

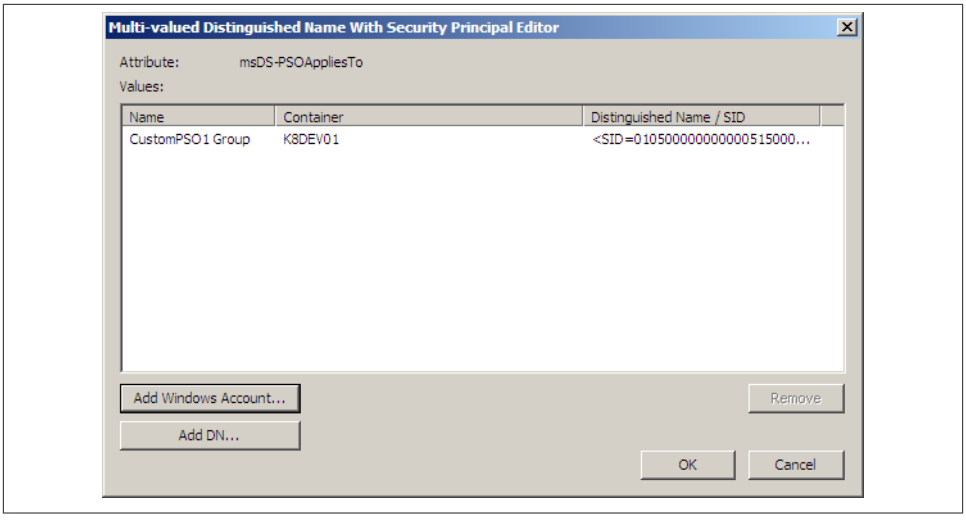


Figure 12-4. ADSI Edit linked value editor

Once you click OK and apply your changes, they will take effect immediately (factoring in replication, of course).

You can remove a user or group from the scope of a PSO by using this same process and selecting the user or group and clicking Remove.

Applying a PSO with ADUC

Microsoft added the ADSI Edit attribute editor property page to Active Directory Users and Computers beginning with the Windows Server 2008/Windows Vista RSAT tools. In order to see the attribute editor in ADUC, check View→Advanced Features. Once you browse to *System→\Password Settings Container* under your domain, you will be able to open a PSO and edit it in the exact same manner as in ADSI Edit.

Applying a PSO with PSOMgr

PSOMgr has a versatile syntax for applying and unapplying a PSO. You can specify a username (`sAMAccountName`), user principal name (UPN), or distinguished name (DN) of the object to apply the policy to. As in the previous examples, we will apply our Presidential Policy PSO to a group called "Presidential Policy PSO Group." The switches we will use to accomplish this are:

- `-applyto (<samname>|<upn>|<dn>)`
- `-pso <psoname>`
- `-forreal`

For example, *psomgr -applyto "Presidential Policy PSO Group" -pso "Presidential Policy" -forreal* is the syntax to apply the PSO in this scenario. After running this command, you should get output similar to the following:

```
PSOMgr V01.00.00cpp Joe Richards (joe@joeware.net) April 2007

Using host: Chicago\K8DEVDC01.k8dev01.brianlab.local
Retrieving PSOs...
Applying PSO cn=Presidental Policy,cn=password settings container,cn=system,
            dc=k8dev01,dc=brianlab,dc=local to object CN=Presidential Policy
            PSI Group,CN=Users,DC=k8dev01,DC=brianlab,DC=local
PSO successfully updated.

The command completed successfully.
```

If you want to unapply a PSO, the syntax is nearly identical. Simply substitute *-unapplyto* for *-applyto*:

```
psomgr -unapplyto "Presidential Policy PSO Group" -pso "Presidential Policy"
       -forreal
```

Viewing the effective PSO

The easiest ways to view the effective PSO for a given user are with ADAC and PSOMgr. Since the `msDS-ResultantPso` attribute is computed, it won't be displayed in a standard listing of all the attributes of a user. Active Directory will only return the value of a computed attribute when it is asked for it explicitly in a search. Additionally, you cannot construct an LDAP filter that filters on the value of a constructed attribute, so it is not possible to search Active Directory for all of the users whose `msDS-ResultantPso` attribute is a certain PSO.

To display the effective PSO for a given user with ADAC, find the user, and then right-click and select "View resultant password settings." This will open the PSO that applies to the user and is effective. If a PSO is not applied to a given user, you will receive an error similar to Figure 12-5 that indicates that the password and lockout policy settings in the Default Domain Policy GPO are in effect for that user.

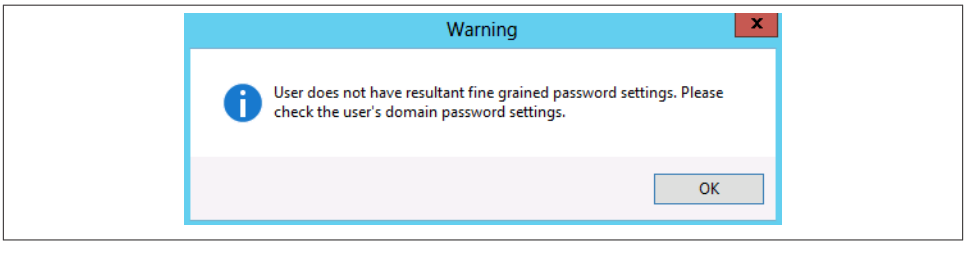


Figure 12-5. Error message when a user does not have an FGPP applied

If you don't have access to ADAC, or need to get the resultant PSOs for a number of users with a script, PSOMgr will be very helpful. Fortunately, there is only one switch necessary for viewing the resultant PSO with PSOMgr:

```
-effective (samname|upn|dn)
```

For example, *psomgr -effective bdesmond* is the syntax to view the resultant PSO for user *bdesmond*. After running this command, you should get output similar to the following:

```
PSOMgr V01.00.00cpp Joe Richards (joe@joeware.net) April 2007

Using host: Chicago\K8DEVDC01.k8dev01.brianlab.local

User: bdesmond | bdesmond@k8dev01.brianlab.local | CN=Brian
Desmond,CN=Users,DC=k8dev01,DC=brianlab,DC=local
Applied PSOs:
  CN=DomPol_k8dev01.brianlab.local,CN=Password Settings Container,CN=System,
  DC=k8dev01,DC=brianlab,DC=local
Effective PSO: CN=DomPol_k8dev01.brianlab.local,CN=Password Settings Container,
               CN=System,DC=k8dev01,DC=brianlab,DC=local
    Type               : Policy Settings Object
    Domain             : k8dev01.brianlab.local
    Policy Precedence  : 20
    DN                 : CN=DomPol_k8dev01.brianlab.local,CN=Password Settings
                         Container,CN=System,DC=k8dev01,DC=brianlab,DC=local
    Name               : DomPol_k8dev01.brianlab.local
    Canonical Name     : k8dev01.brianlab.local/System/Password Settings
                         Container/DomPol_k8dev01.brianlab.local
    Display Name       : PSOMGR: Copy of Domain Policy for k8dev01.brianlab.local
    Lockout Threshold  : 0
    Lockout Duration   : 30
    Lockout Observation: 30
    Min Pwd Age        : 1
    Max Pwd Age        : 42
    Min Pwd Length     : 7
    Pwd History        : 24
    Pwd Complexity     : TRUE
    Pwd Reversible     : FALSE

The command completed successfully.
```

The first part of the output shows each of the PSOs that apply to the given user, and the second part shows all of the details of the PSO that is effective. You can also view the resultant PSO by accessing the Attribute Editor tab of the properties of a user in Active Directory Users and Computers. In order to access the Attribute Editor, you must ensure View→Advanced Features is checked. Additionally, on the Attribute Editor tab you should check Filter→Constructed. This will show the `msDS-ResultantPSO` attribute in the listing.

Delegating Management of PSOs

One of the decisions you will likely need to make when planning your implementation of fine-grained password policies is who will manage the PSOs. It is likely that this will fall into the hands of a few different groups or tools, depending on the organization. One possibility is that the Active Directory team will manage the PSOs, in which case this section is somewhat irrelevant. Another possibility is that the PSOs will be controlled by an information security team. A third is that PSOs will be managed indirectly by an identity management/provisioning system.

In the event that the Active Directory team owns PSOs end-to-end, you likely will not need to perform any delegation of security rights to accomplish the management of PSOs. In the event that a separate team such as an information security group controls PSOs, you will need to delegate the necessary rights. In this case, it is likely that there will be three ways you can handle the delegation:

Delegate access to only application groups
: In this scenario you would strictly employ a design of applying password policies based on security groups, with no exceptions. The information security group would be delegated access to these groups and be responsible for managing the membership of the groups. When a new PSO is deemed necessary, this group will need to engage the Active Directory team to create it. This model creates a system of checks and balances to ensure that growth in the number of PSOs is managed. This model would also require that the Active Directory team be engaged to modify the settings on a PSO.

Delegate access to modify PSOs and application groups
: In this case you would delegate access to modify existing PSOs and also to manage the membership of application groups. The information security group would be able to modify the specific settings on a PSO and control where the PSO is applied. If you feel that you will often need to implement exceptions where PSOs are linked directly to users, this model will enable this to be done independent of the Active Directory team. While chances are that changes to PSO settings will be fairly rare, this model also mitigates the need to engage the Active Directory team for those changes. The system of checks and balances for creating new PSOs still exists here.

Delegate access to create and modify PSOs and application groups
: Here we take the previous model one step further and delegate to the information security team the ability to both modify and create new PSOs. If you have a frequent need to create PSOs, this may be the ideal route. Keep in mind that this removes the checks and balances in the other approaches, so there is a potential for the number of PSOs to grow without bound.

If you are going to use your identity management system or provisioning tool to manage PSOs, you will need to delegate to the account that tool runs under the necessary rights

to manage the PSO application. This delegation may involve allowing the tool to edit the `msDS-PSOAppliesTo` attribute directly, and/or allowing the tool to manage the membership of the application groups for the PSOs.

For more information about implementing Active Directory permissions and delegations, see Chapter 16.

Summary

Fine-grained password policies are an important and long-awaited feature. FGPPs allow administrators to define password settings objects and apply different password and account lockout policies to different sets of users in the domain. PSOs can be applied either to global security groups that users are members of, or directly to users.

Managing PSOs is possible with the Active Directory Administrative Center, Windows PowerShell, or the free PSOMgr command-line tool. The PSOMgr tool offers a number of enhanced features that any administrator who is planning to deploy FGPPs should at least investigate.

Due to the manner in which PSOs are implemented in Active Directory, you will need to consider the management and delegation model of those PSOs as it applies to your organization. Applying PSOs to groups provides flexibility in delegating this responsibility.

CHAPTER 13

Designing the Active Directory Structure

The emphasis of this chapter is on planning the structure of your Active Directory installation. Specifically, we will look at the forest and domain tree layout as well as the organizational unit (OU) structure. While it was extremely common (and often necessary) to design a forest with numerous domains when Windows 2000 came about, that need has largely dissipated. We'll explore how you can reduce the number of domains that you require for Active Directory while gaining administrative control over sections of the Active Directory domain namespace using organizational units. The purpose of this chapter is 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 users, computers, and groups.

When designing a forest, remember that there are often multiple good answers to forest design for any given company. There is no "best" design for all situations. Microsoft has provided great flexibility in what can be done, which can turn around and bite you with indecision about how you should implement AD. It isn't unusual for two engineers to have two very different designs for the same company that are both good for completely different reasons. Simply document all recommended designs and let the decision makers decide together which one will be the best for long-term operations. Overall, the best solutions are usually the simplest solutions. In most cases, you will want to choose single-forest designs over multiforest designs, single-tree designs over multitree designs, and single-domain designs over multidomain designs. The design example shown here is simply that: an example. The company in question could have designed its Active Directory infrastructure in a number of ways, and this is one of them.

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:

- The forest, not the domain, is the security boundary for Active Directory. Anyone with high-level access rights on any writable domain controller in any domain can negatively impact or take control of any other DC or domain in the forest.
- 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.
- Multiple domains cannot be hosted on a single DC. Imagine three child domains under a root domain located in the United States, each of which corresponds to one of three business units. Now imagine that you have a small office of 15 people in Eastern Europe or Latin America with a slow link to the US offices. These 15 users are made up of three sets of 5; each set of 5 users belongs to one of the three business units/domains. If you decide that the intersite link is too slow and you would like to install a local domain controller for these three domains at the remote site, you will need to install and support three separate domain controllers, one for each domain. While this could be virtualized, that is still three additional domain controllers to manage, update, and monitor.
- Too many group policy objects (GPOs) often leads to long logon times, 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. However, if the 10-deep and 5-deep OU structures both contained only two levels with GPOs, they would both incur the same GPO processing.

The Complexities of a Design

Active Directory is a complex service, and designing for it isn't easy. Take a look at the fictitious global company called PetroCorp depicted in Figure 13-1.

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

Now take a look at Figure 13-2, which shows a much more complex hierarchy.

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 (represented by trapezoids) can be applied to various portions of the tree. The following is a discussion of the principles and processes that will help you create complicated designs like these to mirror the complexities in your own organization.

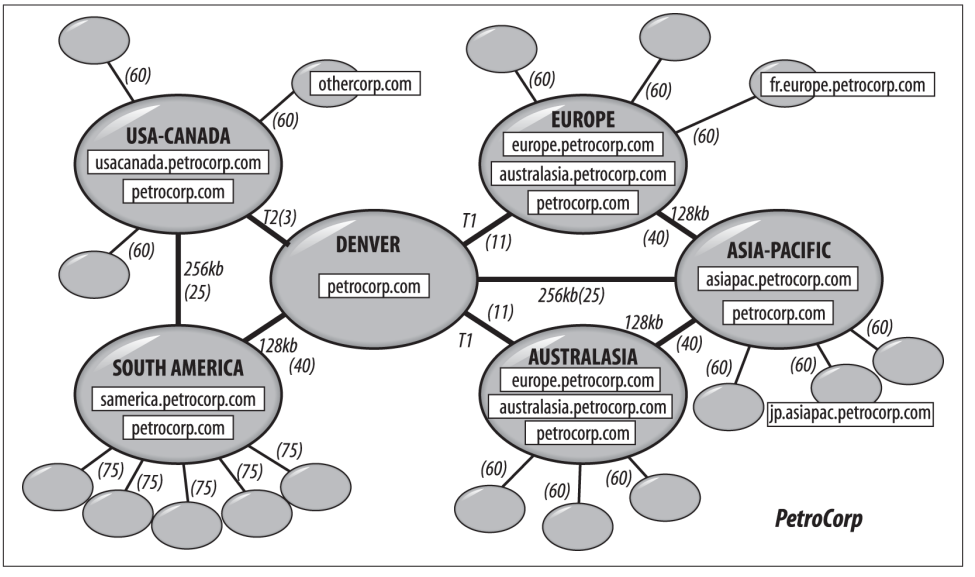


Figure 13-1. The sites and servers of a hypothetical company called PetroCorp

Where to Start

Before you sit down to make your design, you will need to obtain some important pieces of information. At a minimum, you will need:

- A copy of your organizational structure, since this is effectively the document that explains how your organization's business units fit together in the hierarchy.
- 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.
- A copy of the network diagram(s), indicating the connection speeds between the various sites.
- A copy of any diagrams and information on any systems that will need to interface to Active Directory, such as existing X.500 and LDAP directories and major applications, so that you can take them into account.

Once you've gathered the information, you can sit down and plan your design.

Overview of the Design Process

The namespace design process takes place in two stages:

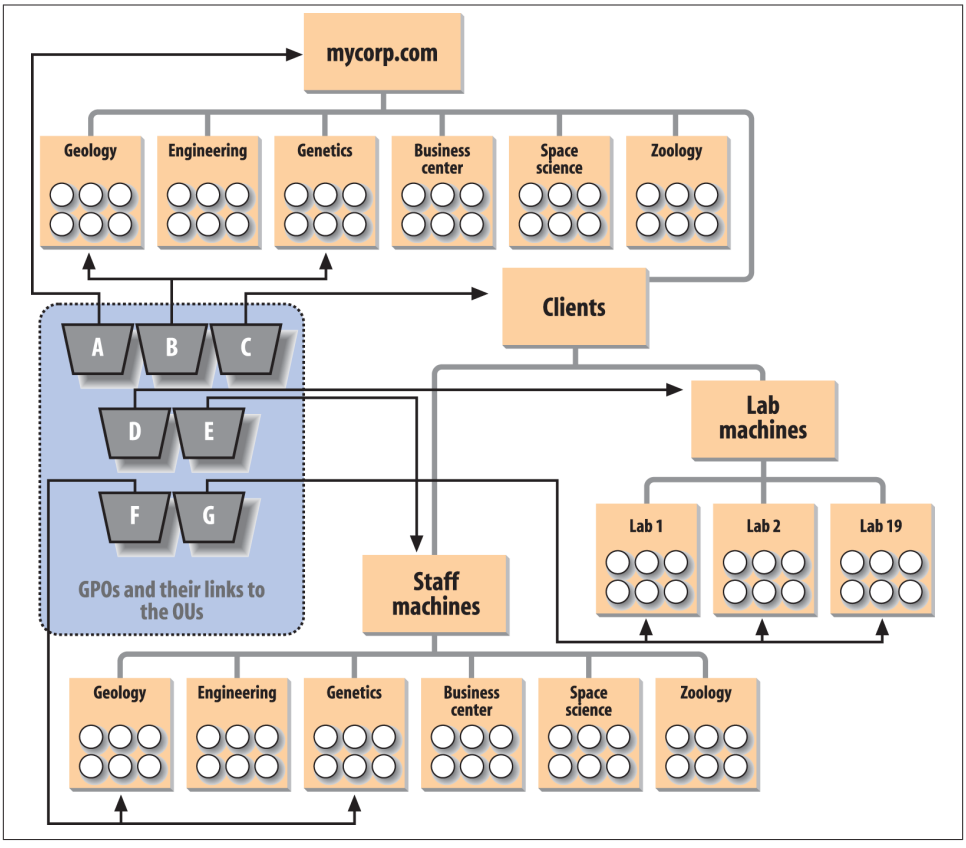


Figure 13-2. A complex domain tree showing GPOs

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 15.

When you are finished with your design, you can implement it 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 importance of a test environment.

When working on the budget for your production forest, include time, money, and resources for the test environment as well. This up-front expenditure will undoubtedly save you considerable backend mistakes and uncertainty on a continual basis.

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 single tree or a series of trees. Before we start, however, let's take a look at our objectives for this part of the design.

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

Represent the structure of your business

When designing your Active Directory environment, you should aim to make it both match the structure of your business, and also be managed by whatever management model your IT organization operates under. Typically, organizations fall into one of three or four common models for managing their IT infrastructure:

Centralized administration
: In this model, your entire IT infrastructure is centrally managed by one team.

Decentralized administration
: In this model, your IT infrastructure is typically managed either locally at each location or on an organizational basis, perhaps by department or division. It is difficult to deploy a common Active Directory forest in this model, as standards are often lax or nonexistent and Active Directory doesn't lend itself particularly well to being administered by a large group from an infrastructure (e.g., domain controllers and forest-level configuration) perspective.

Hybrid centralized/decentralized administration
: In a large organization, this is likely to be the most common model. Certain components of Active Directory are managed centrally, such as the domain controllers and enterprise-wide configuration, while other components, such as objects stored in OUs, are managed by distributed IT staff.

Outsourced administration
: In this model, a third party manages your IT organization for you to some extent. The scope of outsourcing varies from organization to organization, but oftentimes architecture and engineering decisions are retained and not given to the outsourcer to handle.

When Active Directory first came to market in Windows 2000, it was common to deploy multiple domains to mimic the existing NT4 deployment or for various other reasons, and most large companies that deployed Active Directory early on still have this model today. There is nothing wrong with the early best practices, such as empty root domains or regional domains, but conventional wisdom today says that you should endeavor to have the fewest number of domains possible. That said, by no means are we recommending that you rush off to collapse domains created early on unless you can achieve cost or management savings that exceed the cost of this restructuring.

One of the most common justifications that will be made for an additional domain in the forest is a purely political requirement to separate administration. In reality, determined administrators of any domain in a forest could elevate their privileges to the Enterprise Admin level without too much work. Instead of creating a separate domain in this situation at a relatively high cost (with practically no benefit), use organizational units over which you can delegate the necessary permissions.

As you go through the process in this chapter, keep in mind that, especially in large organizations, businesses frequently reorganize. Develop a design that you feel can survive multiple reorganizations without substantial investment in changes to your Active Directory design. Our experience is that designs based more on geography, location, or support models tend to fare better as the organization evolves.

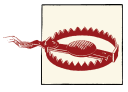
Step 1: Decide on the Number of Domains

Start by imagining that every object is to be stored in one domain. This will give you a much simpler layout for administration. It doesn't matter what the domain is called for now; just label it as your organization's main/sole domain.

Now expand the number of domains that you have by adding other domains that you can give specific justification for. While the number of objects and delegation of administration are not good reasons for creating new domains, there are still a couple reasons that would require more domains:

- The need to isolate replication
- A requirement for disparate Kerberos domain policies

If you can match either of these criteria, write down a new domain for that area.



There are good reasons to add domains to the design, but there are also good reasons to not add domains. You should weigh these against the requirements listed previously to determine which are most important.

Some reasons to not deploy an additional domain include:

Forest complexity
: Each domain requires more DCs, more policy management, and additional cost.

Application complexity
: Applications may only be able to search a single domain for information. Many Unix- and Java-based LDAP applications are generally written to have a single simple hierarchy and may not be able to be configured to work with a multidomain forest without serious work.

Isolated replication

Although we'll discuss replication in more depth in the next chapter, you should begin to consider it from a high level at this point, as it can have a substantial impact on the domain design you settle upon. Many large organizations have elected to deploy domains in a regional fashion that matches major divisions in WAN topology. Such models often include domains such as Americas, AsiaPacific, and Europe or EMEA (EMEA is a common acronym for Europe, Middle East, and Africa). The reasoning for this is to segment replication at a high level. Environments with large domains cannot always bear the replication overhead of replicating all of the contents from an Americas domain to a remote site in Asia, for example.

Since Windows 2000, replication has improved greatly (with respect to compression, in particular), and WAN links have gotten faster in some regions. As you explore design options, you will need to consider whether or not your WAN could be impacted by the replication overhead of a large domain and if limiting this replication would be beneficial. Keep in mind that even if you segment your domains on a regional basis, there will still be some interregional replication for the Global Catalog.

Unique domain policy

In Chapter 11 and Chapter 15, we explain the basics of group policies and how to properly design them. For now, the important thing to understand is that policies are Active Directory objects that reference a number of settings that can be applied to users

or computers. These settings are things like a fixed desktop, a certain core set of applications, the ability of users to perform a shutdown, and so on. If you create an organizational unit called Finance and then decide that it needs special settings, you can create a GPO and assign it to the OU. Then the computer settings and user settings in the GPO will be applied to all computers and users under the Finance OU.

We now need to look at what settings have to be applied on a domain-by-domain basis. Here's a list of what types of settings can be set only on a domain-wide basis:

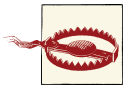
- Kerberos policies
- Encrypted filesystem (EFS) recovery policies
- Public key encryption policies

If a special department or geographical area needs special encryption, security safeguards, certificates, and so on, you may need a separate domain for it.

Domains that are not operating at the Windows Server 2008 or better functional level can only have one password policy per domain. For more information about fine-grained password policies, see Chapter 12.

Final notes

You now should have your first draft of the list of domains that you think you will need. There is one more very important point on this subject. Domains are very inflexible and unforgiving, and due to the fact that you can host only a single domain on a domain controller, each domain you add means at least one more domain controller you have to support.



For fault tolerance, you should always deploy new domains with at least two domain controllers. If you only have a single domain controller for a given domain and the domain controller fails, you will be forced to restore from a backup. This is easily avoided by deploying a second domain controller.

Depending on how many domain controllers you would have to deploy for a domain, you can greatly decrease your total cost of ownership (TCO) for Active Directory by limiting the number of domains you support. In general, we recommend that you start with a single-domain design for any new Active Directory forest unless you have a compelling technical reason not to.

Step 2: Design and Name the Tree Structure

Now that you have the domains listed, you need to consider what sort of hierarchy to put them in. It is easiest to start with one domain, the one that will become the forest root.

Choose the forest root domain

There are two ways to go about choosing the forest root domain. The first model is to deploy an empty root domain. The empty root simply contains a small number of domain controllers and accounts for the Active Directory administrators for the forest. The empty root is the most common model you're likely to see in large environments that have had Active Directory deployed for some time.

The security concerns that led many organizations to deploy an empty root have since been proven to be a nonissue. While the empty root model is perfectly valid, you should not choose to deploy a forest with an empty root solely with the goal of protecting the enterprise-wide groups (such as Enterprise Admins and Schema Admins).

The second model is to simply choose a domain that you know will be required for the lifetime of the forest as the forest root domain, which can never be removed. If you are deploying a single-domain forest, this decision is simple. If you are deploying multiple domains, you should keep in mind your namespace requirements as you design the hierarchy, such that the forest root domain is actually at the root of the tree.

If you are planning to deploy a single-domain forest—the model that we generally recommend—there is no reason to deploy an empty root simply because it is an option. Making your sole domain the forest root is perfectly acceptable.

Design the namespace naming scheme

As each domain has a DNS name to identify it, you need to consider what names you are going to choose. You can use any of the RFC 1123 standard characters:

- A–Z
- a–z
- 0–9
- - (dash character)

Microsoft's DNS server implementation supports a wider range of characters, such as the Unicode character set, but if you need compatibility with other DNS flavors, be very careful allowing these.

There are currently two schools of thought on how to pick the DNS names for your Active Directory network: *root zone* or *subzone*. The root zone method says that you name your root Active Directory domain based on the root zone for your organization. For the Coho Vines forest, this would be *cohovines.com*. The subzone method suggests that you pick a new subdomain from your root zone and make that the base of your Active Directory namespace. For Coho Vines, this could be *ad.cohovines.com*.

If you choose the root zone method and wish to have a non-Windows DNS server at your root, you will need to either enable dynamic updates or manually register a number of records in the DNS servers, as discussed in Chapter 8. If you choose the root zone method and wish to have a Windows DNS server at your root, you will need to migrate your existing entries, if you have any, to the new DNS servers. Both methods are fine, but they require configuration or migration at the root.

A less invasive procedure would be to choose a new subzone for your Active Directory network and run your network from that. With this setup, you still have two choices, but they are less disruptive to any existing structure and don't affect the main root zone. Arguably, the easiest solution is to let two servers on your network run Windows DNS Server and manage this DNS zone. This allows you to have a root that doesn't allow dynamic updates and a subdomain that does. In this case, your existing main root zone DNS servers would contain a DNS delegation for the subzone. The subzone would be delegated to your Active Directory DNS servers. The alternative would allow a non-Windows DNS server to manage the zone.

Another common extension of the root zone method is to choose a parallel namespace, such as *cohovines.net,* for your Active Directory network. This allows you to have a root-level namespace and not need to migrate existing DNS entries or manage a split-DNS model if *cohovines.com* is also your external namespace.

To continue with the planning process, start with the forest root: draw a triangle on the paper and assign a DNS name to the domain, writing the name inside or beside the triangle. You should choose the name very carefully, for two reasons. First, while renaming a domain is possible beginning with Windows Server 2003 Active Directory, it is a highly invasive process and is not generally recommended unless absolutely necessary. Second, you can never remove the forest root domain from Active Directory. If you need to change its name, you must destroy your entire forest and start again.

Create additional trees

Having created and named your forest root, you need to consider your other domains. If you have two distinct business units or companies, some analysts may think that they will require noncontiguous names to match the structure. This means that you will need two trees coming from a domain root. Think long and hard about this decision—multiple domain trees in a forest add complexity and often cause support and application confusion for no real benefit. Some common examples that people will use to indicate

a perceived need for multiple trees are different email address suffixes, different UPN suffixes, resource separation, and security separation. However, separate trees aren't actually required to implement any of these requirements.

If you still conclude that you need multiple trees, draw all the other tree root domains that you think you will need as separate triangles at the same horizontal level on the paper and assign them valid DNS names. These domains are all tree root domains, but they should not be confused with the forest root domain.

A real-world example of this scenario is the Microsoft brand name and the MSN brand name (Microsoft owns MSN). Both *msn.com* and *microsoft.com* could be separate trees in the same forest. They couldn't be in the same tree without giving them a hierarchical link, i.e., *msn.microsoft.com*. However, the only real benefit for having an *msn.com* and a *microsoft.com* in the forest instead of an *msn.microsoft.com* is how it all looks on a PowerPoint presentation.

If we think that Coho Vines's Finance Department needs a separate domain, we will make a subdomain and call it *finance.cohovines.com*. Within Active Directory we could make *finance.cohovines.com* a separate tree in its own right, but since hierarchical and transitive trusts exist throughout a forest, we would gain absolutely nothing by doing this. The only differences come in choosing Finance to be a new domain (which we did) or a new forest in itself. Making it a new tree gains us absolutely nothing.

Create additional forests

So far, we've been considering domains that will exist in the same forest, but you may have business units that will require two entirely separate forests. How do you know if that is the case? There are a number of common scenarios for this.

If you have business units in an organization that are independent and in fact wish to be isolated from each other, then you must not combine them in a single forest. If you simply give each business unit its own domain, these business units can get the idea that they are autonomous and isolated from each other. However, in Active Directory, this level of isolation can be achieved only through separate forests. This is also the case if you need to comply with regulatory or legal isolation requirements.

The first and most common reason for this design may be political: certain business units may decide that they want to be as autonomous as possible. It may be that, politically, the Finance Department has to be completely separate, so you end up making a second forest with *finance.cohovines.com* as the second forest's forest root domain. In effect, you are treating this business unit as a separate, autonomous, and noncontiguous part of the tree.

Another reason that you may need two forests involves having two businesses that must be separately maintained for regulatory or other legal reasons. Similarly, if you have a business that may be divested or sold in the future, it will be much easier to separate if this business exists in its own forest.

The third reason is one born out of necessity. Remember from Chapter 2 that certain aspects of a namespace are forest-wide in scope. If you want to isolate a separate schema or configuration partition, your only solution is to create a separate forest.

The fourth reason, which is becoming more common, is the need for a separate forest for Microsoft Exchange. Exchange is very tightly integrated into Active Directory and does not always allow for a clean separation of duties through delegation. Additionally, Exchange can have management challenges with multidomain deployments in a single forest. These two considerations can often result in a single-domain resource forest being implemented for Exchange, with trusts configured to the appropriate domains in the primary forest or to the primary forest itself.

Finally, a fifth reason is to separate specific users, servers, or applications. It is not uncommon to deploy a dedicated forest in a perimeter network and join those servers to a domain in the primary forest. A one-way trust from the primary domain(s)/forest to the perimeter network forest enables internal users to access applications in the perimeter network, but it does not allow servers in the perimeter network to access internal resources. A common extension of this model is a specific forest for external users (such as business partners or customers) that need to access an application.

If any of these reasons apply, you need to create a second forest root domain and give it a unique DNS name, as you did for the first forest root domain. In effect, you need to separate your designs and design each forest individually. The best thing to do now is to figure out how many forests you need and which domains from your list are going to be the forest root domains. Once you have determined this, you will name these root domains and then use a separate piece of paper to draw each forest. Maintain separate lists of domains for each forest. You're now creating *x* designs, where *x* is the number of forests you have.

There is one other important point that you need to be aware of. While domains and trees in a forest maintain automatic trust relationships, it is possible to set up manual trust relationships with domains external to a forest. You can therefore set up manual trust relationships between forests. These relationships can be one-way trusts (A trusts B but B does not trust A) or two-way trusts (A trusts B and B trusts A).

While there is transitive trust between all domains in a forest, it is not uncommon to create *shortcut trusts* in large multidomain forests. Shortcut trusts improve authentication performance in scenarios that require users in one child domain to access resources in another child domain. Without a shortcut trust, authentication traffic must walk the domain tree to get to the other child domain. Shortcut trusts create a direct path.

The first option allows other domains that are members of another domain tree in a different forest or that do not support Kerberos authentication to have limited access to a domain in the forest. Only resources in that domain will be visible; no other resources in the tree will be available for access. The second option is to create a forest trust between two forests. With a forest trust, all of the domains within each forest will transitively trust each other.

Arrange the subdomain hierarchy

You now have a forest root domain with a valid DNS name. You may have other domains that act as the roots of separate trees in the same forest; you may even have extra forest root domains representing separate forests entirely. Now you need to lay out the domain tree hierarchies. If you have a number of remaining domains listed on your sheet of paper from step 1, these are the subdomains that will form your domain-tree hierarchy.

Start with the first forest. Representing each domain with a triangle on the paper, arrange the domains in a hierarchical fashion beneath one of the domain tree roots in the forest. Name each domain appropriately, according to its position in the hierarchy. Repeat this process for all domains in this forest, then move on to the next forest (if necessary) and repeat.

For example, if we have *cohovines.com* as a tree root, and the North America, Europe, and Asia-Pacific regions all need separate domains, we can call them *northamerica.cohovines.com*, *europe.cohovines.com*, and *asiapacific.cohovines.com*. If the Asia-Pacific region needs a separate domain for Japan, we'll arrange this domain beneath *AsiaPacific*, as *japan.asiapacific.cohovines.com*.

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.

When designing the internals of a domain, you need to consider both the hierarchical structure of organizational units and the users, groups, workstations, and servers that will reside within those organizational units. Let's look at each of those in turn.

Moving forward, 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.

Step 3: 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 your requirements for a hierarchical set of administrative permissions using organizational units instead.

OUs are the best way to structure your data because of their flexibility. They can be renamed and easily moved around within 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 OU structure that you will instantly recognize makes managing it much more comfortable than it would be with every user and computer in one OU.

To delegate administration
: Active Directory allows you to set up a hierarchical administration. If you have three branch locations, 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 for yourself: political necessities, organizational models, and so on. Active Directory really won't care how you structure your data.

To apply policies to subsets of your users and computers
: As group policies can be applied to each individual OU in the hierarchy, you can specify that different computers and users get different policies depending on where you place them in the tree.

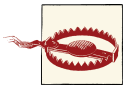
  For example, let's say that you want to place interactive kiosk machines in the lobby of your office and allow people to interact with whatever applications you specify, such as company reports, maps of the building, and so on.

With Active Directory, if you apply group policies to a certain OU hierarchy, you can guarantee that any computer and user accounts that you create or move to that part of the tree will always implement the kiosk lockdown policies you have designed.

When creating organizational units, you need to ask:

- How will the organizational units be used?
- Who are the administrators and what sets of administrator permissions should they have?
- What group policies will be applied?

The hierarchy should organize information in a manner that enables your administrative model, enables effective use of group policies, and allows you to delegate administration to various parts of the tree.



You should not nest user or computer accounts in an organizational unit structure in such a way that the group policies that apply to the accounts create a slowdown during the logon process. Microsoft recommends nesting no more than 10 organizational units deep, but in fact, to a much greater extent, it's the application actions of group policies that you need to consider when designing your OU structure. This helps prevent slowdowns during boot (policies applied to the computer account on boot up) and logon (policies applied to the user account on logon).

Nesting OU structures excessively also has a tendency to lead to substantially increased administrative overhead due to the additional complexity of the model.

Recreating the business model

The easiest way to start a design is to consider the business and administrative models that you sat down with when creating these designs. You now need to recreate that structure in Active Directory using OUs as the building blocks. Create a complete OU structure that exactly mirrors your business model as represented by that domain. In other words, if the domain you are designing is the Finance domain, implement the finance organizational structure within that domain. You don't create the entire organization's business model within each organizational unit; you create only the part of the model that would actually apply to that OU. Draw this structure out on a piece of paper. Figure 13-3 shows a sample structure that matches the business's organization directly. For simplicity, we've expanded only the Finance OU here.

Once you have drawn an OU structure as a template for your Active Directory hierarchy within the domain, you can begin to tailor it to your specific requirements. The easiest

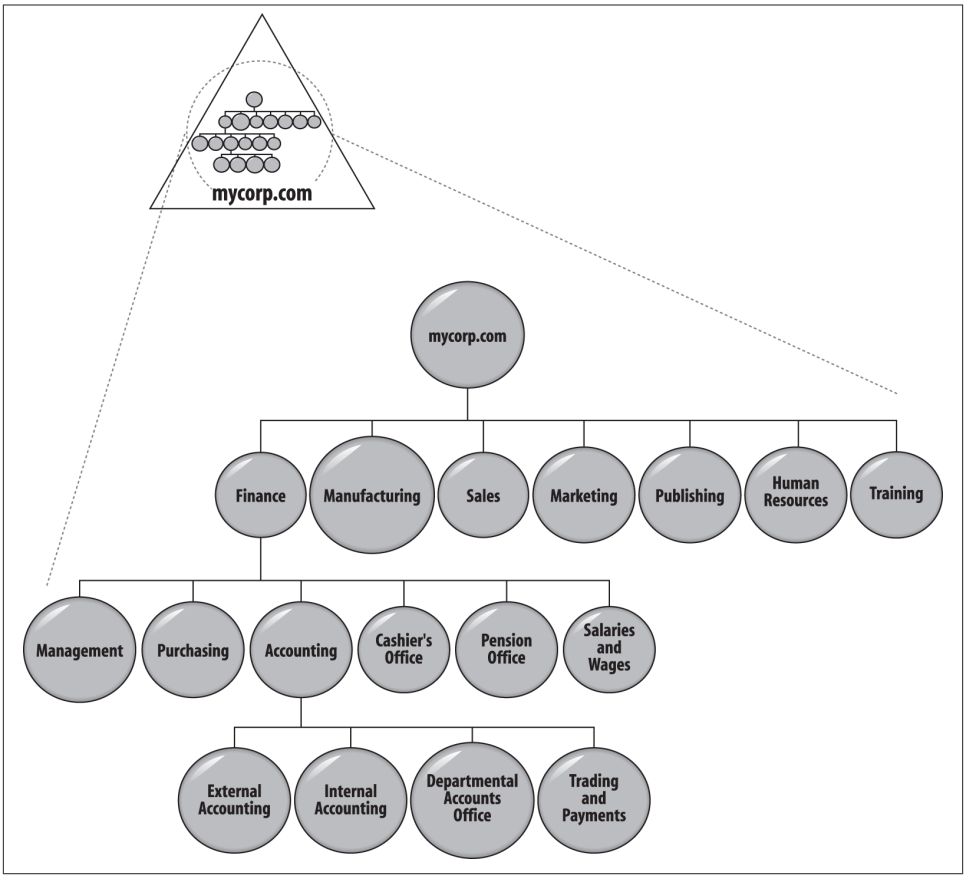


Figure 13-3. The Mycorp domain's internal organizational unit structure

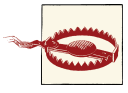
way to tailor the initial OU design is to consider the hierarchy that you wish to create for your delegation of administration.

Delegating full administration

First, identify any areas of your hierarchy where you need to grant administrators autonomous access over their branch of the tree. Each OU should have at least two administrators who will look after that organizational unit. These administrators will take care of the structure below the organizational unit, creating whatever OUs, users, groups, and so on that they desire. They will not, however, have administrative access to any other part of the tree.

You need to note two pieces of information about each of the organizational units that you identify:

- Who will be the administrators?
- Which branch of the tree will they administer?



You must ensure that delegated administrators take responsibility for their actions and can be held accountable. This cannot be stressed enough. It is entirely possible for an administrator of a low-level organizational unit to make changes that affect other people.

Delegating other rights

Having noted these two pieces of information for all organizational units that need full administrative access, you next need to identify those organizational units that require some users to have a more restricted set of permissions. You may want to set up account administrators that have the ability to create and delete user accounts, and to set passwords and account details. We're interested in rights only in general terms at the moment, so just note the following:

- What the access rights are
- Which branch of the tree the access rights will be applied to
- Which users or groups (in general terms) will have these access rights

It is possible to set access rights of any sort down to the individual property level on a specific object if you require. That means you can allow a user named George Washington to change the password or telephone number of a user named James Madison (and only that user) if you wish. Obviously, the more granular the access, the more complex things can get, especially since permissions are inherited down a tree by default. To make things easier, Microsoft provides a simple Delegation of Control Wizard that allows you to set these access rights in a fairly easy manner. Active Directory permissions are covered in much greater depth in Chapter 16; all we're concerned with at this stage in the design is delegation of control at the organizational unit level. From experience, we can tell you that assigning access rights at the organizational unit level is always much simpler to manage than tracking permissions to individual objects and properties.

Perhaps the most important thing to remember when designing your organizational unit structure is to make it consistent and repeatable. If you are creating an OU structure by location, develop a script to create that standard structure (and any applicable permissions delegations) for each of your locations. Managing OU structures that are independently designed and managed will quickly become extremely difficult, if not impossible, to do efficiently.

Step 4: Design the Workstation and Server Naming Conventions

You now have one or more forests of domain trees, as well as an OU structure. You can now consider the naming convention for the servers and workstations.

While we are considering the naming convention here, the exact placement of machines in Active Directory is covered in Chapter 15, where we discuss designing GPOs. That is because GPOs can impact clients based on machine location.

Each client workstation or server in an Active Directory network must have a computer account somewhere in the forest to let users log on via that client. When a machine is added to a domain in a forest, the computer account is created in Active Directory, and a trust relationship is established between the client and the domain so that the client is recognized as a valid member of the domain.

Where a client is placed in the forest determines part of the name. Member servers are usually placed in the domain that hosts most of the users that use the server, and DCs are located by their very nature in the individual domains that they host. Clients can be placed anywhere, but they are usually placed in the domain that the primary users of that client will normally log onto.

All hosts are named *<computer>.<domain>*. For example, a server called *SRV01* in the *cohovines.com* domain would usually be called *srv01.cohovines.com*; a server called *SRV02* in the Europe domain would usually be called *srv02.europe.cohovines.com*.

It isn't normally discussed, but the DNS domain name of the machines in a given domain does not strictly need to match the Active Directory domain name.

This is one example of a disjoint namespace, and it is a supported configuration by Microsoft. This type of configuration is sometimes found in larger Enterprise-class organizations that have complex distributed DNS configurations. You may find, for example, a server with the name *srv01.detroit.michigan.us.cohovines.com*, which is a member of the AD domain *northamerica.cohovines.com*.

Deploying Active Directory does not force you to change the names of any existing hosts that you have. However, if you are planning to consolidate a number of separate domains, you need to ensure that all hostnames are unique within the domains they exist in, or throughout the entire company if WINS is being used with full mesh replication.

If you don't already require a consistent naming scheme for your clients and servers, now is the time to create one. Fully qualified domain names must be unique across the entire forest; this is achieved by appending the domain component onto the computer name. That leaves you to worry about the prefix, meaning that computer names have to be unique only domain-wide.

To maintain backward compatibility, names cannot be longer than 15 characters. This is because Active Directory still has some legacy support for NetBIOS names, and the hostname that you choose will be incorporated as the NetBIOS name on the client. NetBIOS names are limited to 15 characters.

You need to work out a forest-wide naming convention, determining how you will name the clients and servers within the 15-character limit. The convention is up to you, but we supply some samples later in this chapter when we look at sample designs.

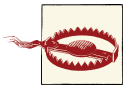
Step 5: Plan for Users and Groups

Before starting this section, we must make clear the distinction between groups and OUs. OUs are containers for objects that allow the objects to be represented in a hierarchical structure within a domain. Placing objects in such a hierarchy allows delegation of administration to those organizational units on a selective basis, as well as application of group policies. We've seen all this already.

Groups, on the other hand, have only users or computers (or other groups) as members and can be used in assigning permissions or rights to the members collectively. Let's say that we have 50 users contained in an organizational unit called Finance, and the users are also members of a group called Finance Users. When we want to grant these 50 users read permissions to a share or restricted access to certain parts of a domain's hierarchy, we assign the permissions to the Finance Users group. The fact that the users are in the Finance OU makes no difference when you wish to assign permissions to objects contained inside the OU. However, if we wish to delegate someone to have permission to manage those 50 accounts, we place the administrative delegation onto the OU. Here, we'll be talking about how to effectively plan for user accounts and the groups to which those users belong.

Naming and placing users

When you are planning for user accounts, the only thing you really have to worry about is the username or user identifier that the client logs on with. Each username (the `sAMAccountName` property of a user object) must be unique throughout each domain. If you have decided to delegate administration within your organization, and you allow delegated administrators to create user accounts, you need to create a naming convention that each administrator will adhere to so that unique usernames are generated for all users throughout your forest.



If you choose to allow delegated administrators the ability to create user accounts, consider the security implications carefully. When user accounts are created in a distributed fashion, there is generally no audit trail, well-established ownership chain, or account lifecycle applied to that user account. We strongly recommend that you look to centralize user creation through a request process or via an automated toolset to establish and maintain these types of security controls.

Another name that you assign to all Active Directory users is the user principal name (the `userPrincipalName` attribute of the user object). This property looks like an RFC 2822 email address, i.e., *username@cohovines.com*. In reality, this property is not the email address but is a unique identifier for the user across the entire forest. The value must be unique as it uniquely identifies a user across an entire forest. So, while having a user with a `sAMAccountName` value of *George.Washington* in *cohovines.com* and in *executives.cohovines.com* is perfectly valid, the UPNs for each user account must be unique.

Often, UPNs are created by simply appending an @ symbol and the domain onto the end of the username. This ensures uniqueness because the username is unique in the domain, and appending the domain forces a unique forest-wide UPN. This makes *George.Washington@cohovines.com* and George.Washington@executives.cohovines.com the UPNs for the two user accounts in the preceding example.

However, while it is conventional to construct the UPNs in this way, you can in fact make the UPN of a user anything you wish as long as it follows the format specified in RFC 2822. We could, for example, append *@cohovines.com* to all our users, eliminating the need to rely on domains at all. If we do this, though, we should ensure that all of our usernames (`sAMAccountName`) are unique forest-wide. In the previous example, we wouldn't be able to have two users with the username *George.Washington*. For such a scheme to work, a central database or allocating authority needs to be set up to uniquely generate and allocate names. If this database or authority can generate unique usernames via a reliable algorithm, you can make use of a much simpler UPN.

Just because we chose *@cohovines.com* as the suffix does not mean we are limited to a forest or domain name. We could just as easily have chosen *@contoso.com*, which has no relation to the domains or the forest. The UPN simply has to be unique for every user in the forest.

Just as the suffix doesn't have to match the forest or domain name, the username portion doesn't have to match the `sAMAccountName`. In fact, you can create a user without specifying a `sAMAccountName`. This will autogenerate a `sAMAccountName` that looks something like `$KJK000-H4GJL6AQOV1I,` which really isn't a friendly logon name. However, you could then set the UPN to be *George.Washington@cohovines.com* and that could be used to log on.

UPNs are very important. Imagine a user sitting down at a client anywhere in a forest and being presented with the logon dialog box. Here, the user can type in her username (`sAMAccountName`), password, and domain, and be authenticated to the forest. However, it is perfectly valid to authenticate with the UPN. If the user, presented with the same logon dialog box, instead types a UPN in the username field, she will no longer need to specify a domain.

In other words, a UPN and a password are all that is needed to authenticate to Active Directory. This makes sense since the UPN is unique forest-wide; apart from a password, nothing else should be needed. You now should be able to see that even with a very large and complex set of domains in a forest, you can use a simplified UPN form that does not contain a domain component and then simply instruct users to log on with a UPN and a password. This means that users never need to care about what domain they are in.

Many organizations choose to use their users' email addresses as UPNs. The benefit to this strategy is twofold. The first benefit is that users can be instructed to log in with their email addresses and thus are required to remember one less piece of information. The second benefit is that email addresses are unique identifiers, so the UPNs will be too.

Your choice of where you place the user accounts in each domain's hierarchy is really affected only by who is to administer the accounts and what GPOs will be applied on the OUs the accounts are in. Other than that, it makes little difference.

Naming and placing groups

Groups need unique names, too, so a naming scheme for groups should also be laid out in the design. Where you put groups is less important. In effect, groups can go almost anywhere in the hierarchy. The GPOs that determine your placement of users, for example, do not apply to groups.

Much like users, groups also have a `sAMAccountName` field that must be unique within the domain. If you are delegating administration such that delegated administrators can create groups, we recommend that you devise a naming convention that establishes the department or organization that owns the group in the name. For example, you might prefix all group names with a two- or three-letter code that refers to the group's owner.

If possible, we highly recommend that you consider an external tool for managing group objects if you are going to delegate the ability to create and manage groups. This will allow you to establish group purpose, ownership, audit trails, and lifecycles for groups. This helps limit the sprawl of groups in the directory that have no known purpose or owner. Microsoft Forefront Identity Manager (FIM) is one example of a tool that can help meet this recommendation.

Other Design Considerations

In many cases, you may need to revise your designs a number of times. Certainly GPOs will make a difference as to how you structure your user and computer objects, so we do not expect that one pass through the 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 would 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 moving 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 ideal.

Apart from GPOs, which we cover in Chapter 11 and Chapter 15, 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. You might also want to use the Distributed Filesystem (DFS), which allows you to organize disjointed and distributed shares into a single contiguous hierarchy. When you reference a share held by the DFS, the DFS uses the Active Directory site topology 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. If you're thinking about this, 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 it takes to replicate to all DCs throughout the organization. When considering which objects to add, rule out objects with very short life spans. Dynamic data—that is, data with

a relatively short life span—is more suited for storage in an application partition or an AD LDS instance, or possibly through the use of the dynamic objects feature.

- Any object that you include will have attributes that are held in the Global Catalog. For every type of object that you seek to store in Active Directory, check the schema definition for that object to find out what attributes will be stored in the Global Catalog.

Design Examples

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

We used a number of criteria to create these organizations:

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

Tailspin Toys

Tailspin Toys is an organization that employs approximately 3,000 people across North America. There are approximately 4,000 client computers and several hundred servers spread across the Tailspin Toys locations. Office workers are located in a number of cities across the United States and Canada, as well as in home offices. Manufacturing operations take place in Mexico. The IT Department located in the San Francisco headquarters office manages IT assets across the entire organization.

Step 1: Decide on the number of domains

Given the small size of the organization, and connectivity requirements, Tailspin Toys elects to deploy a single-domain Active Directory forest. There are no requirements identified where a separation of replication would be necessary. Likewise, there are no concerns that require any resources to be separated into a separate forest.

Step 2: Design and name the tree structure

This domain/forest will be called *tailspintoys.com* with a NetBIOS name of *Tailspin*. Given the simple design, there is no need to worry about tree structures or naming beyond the name of the root domain.

Step 3: Design the hierarchy of organizational units

Tailspin Toys manages almost all of the objects in Active Directory centrally. Since there is limited need for complex permissions delegations, Tailspin Toys focused on Group Policy when designing its OU structure. After reviewing the organization's needs, a location- and object-based OU structure was developed. This allows policies to be targeted to physical locations as well as specific types of objects (e.g., desktop computers versus laptops). Figure 13-4 shows the top-level OU structure for Tailspin Toys.

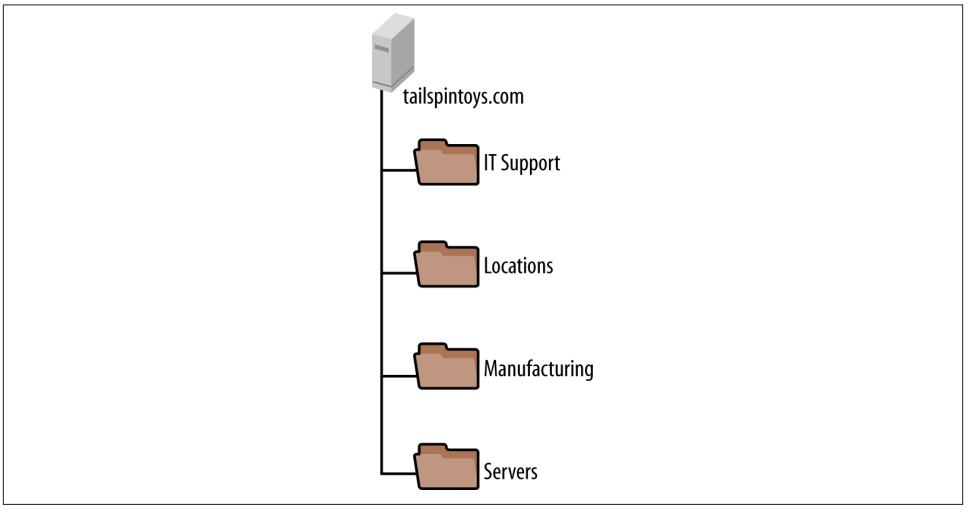


Figure 13-4. Tailspin Toys top-level OU structure

Due to the business-critical nature of machines that support manufacturing activities, a separate Manufacturing OU structure was established for shop-floor computers to isolate them from management of end user machines.

Figure 13-5 shows the OU structure under the Locations OU for the San Francisco office, as well as the parent structure. Organizing the machines hierarchically by geography allows the IT Department to apply group policies targeting users globally, regionally, or at a per-location level. The IT Department has also assigned a location code to each site to identify the site on the network. For simplicity, location codes are a combination of the three-letter airport code for the city as well as a two-digit number, in case there are multiple offices in a given city.

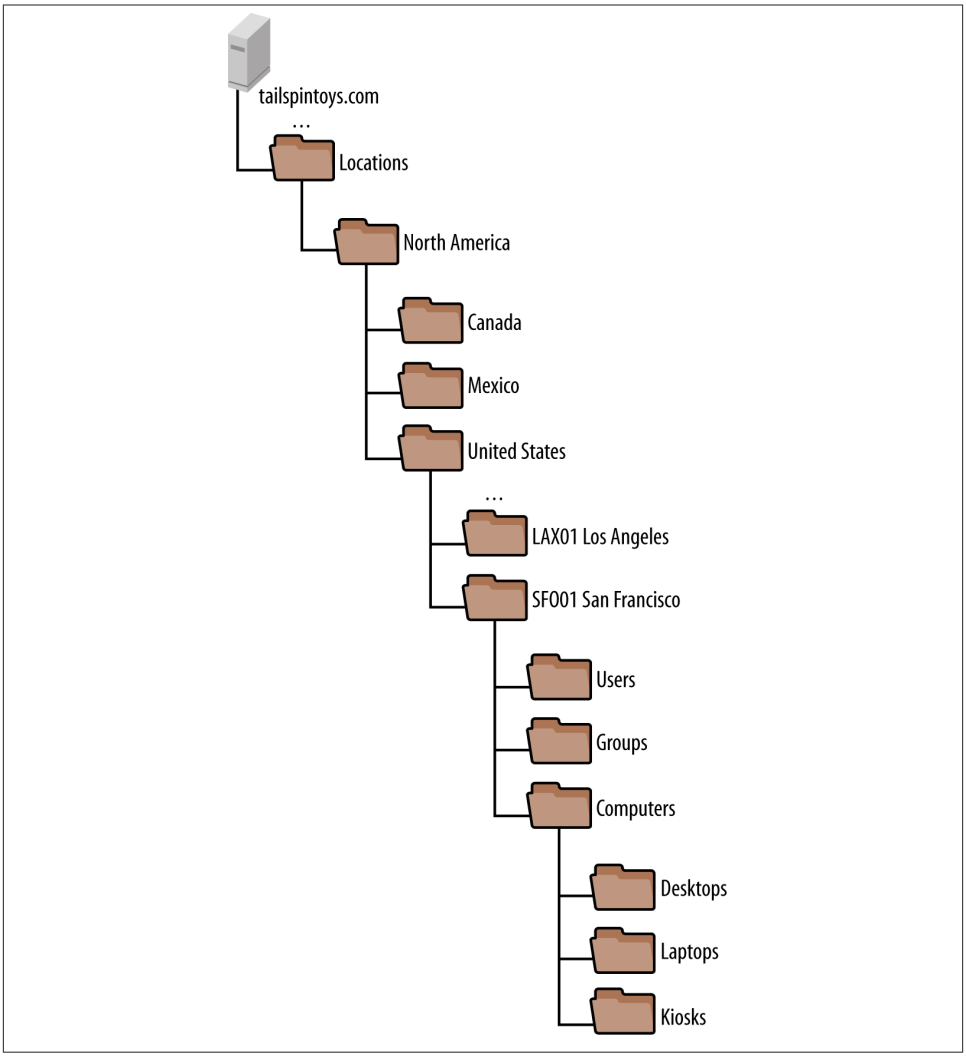


Figure 13-5. Tailspin Toys Locations OU

To keep management straightforward, a similar structure was developed for the Manufacturing OU. Since the Manufacturing OU will only contain machines used to support shop-floor activities, a consolidated version of the structure in Figure 13-5 was developed, as shown in Figure 13-6. Manufacturing machines are simply placed in the site's OU.

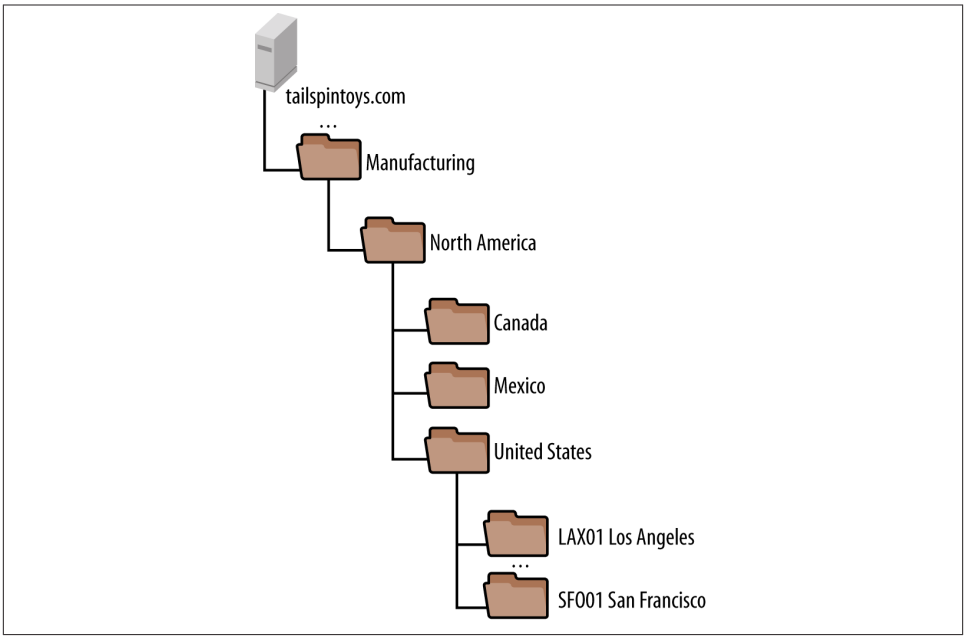


Figure 13-6. Tailspin Toys Manufacturing OU structure

Once the OU structure was finalized, a permissions delegation plan was developed. Based on Tailspin Toys's centralized IT organizational model, there are only two sets of permissions that need to be delegated in Active Directory:

- Password reset and account unlock for helpdesk analysts
- Join computers to the domain for desktop technicians

Both of these tasks were easily delegated using the Delegation of Control Wizard in Active Directory Users and Computers.

Step 4: Design the workstation and server naming conventions

Since Tailspin Toys places its workstations in OUs based on the physical location of the machine, the type of machine (e.g., desktop or laptop), and the function (client computer, kiosk, or manufacturing machine), it decided to incorporate all of these data points in its naming convention. Additionally, based on input from desktop technicians in the field, the location (e.g., office number or cubicle) of the machine at an individual site is incorporated as well.

Tailspin's naming convention is documented in Figure 13-7. The naming convention documents each possible character within the 15-character maximum limitation Active Directory imposes.

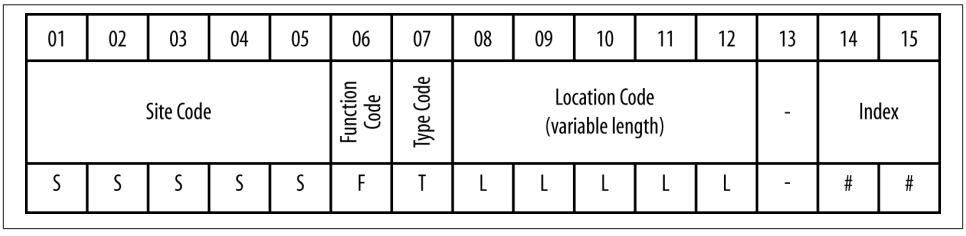


Figure 13-7. Tailspin Toys computer naming convention

The fields are:

Site Code
: This is the five-character code assigned to each physical location. Tailspin uses these codes across all of IT, including network devices, servers, desktops, and the OU structure in Figure 13-5. The first three letters are the airport code for the nearest major airport, followed by a two-digit number identifying the specific location within a given city.

Function Code
: This letter identifies whether the machine is a client computer (C), a kiosk machine (K), or a manufacturing machine (M).

Type Code
: This letter identifies whether the machine is a PC desktop (D), PC laptop (L), Macintosh desktop (M), or Macintosh laptop (P).

Location Code
: This field allows for up to five characters that can be assigned by the local desktop technician. Desktop technicians typically encode the office location, cubicle, shop floor column number, or similar information in this field.

Index
: This is a numeric field, beginning with 01, used to disambiguate multiple machines in a given location.

Based on the correlation of the naming convention to the OU structure, a script is developed to automatically move any machines in the default Computers container to the appropriate OU. The script is scheduled to run every five minutes. This alleviates the need for IT to manually move computers in the directory.

The server naming convention illustrated in Figure 13-8 is straightforward and similar to the desktop naming convention.

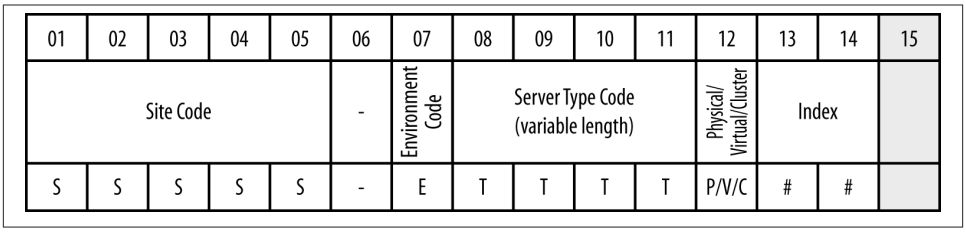


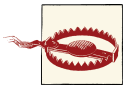
Figure 13-8. Tailspin Toys server naming convention

The fields are as follows:

Site Code
: This is the five-character code assigned to each physical location. Tailspin uses these codes across all of IT, including network devices, servers, desktops, and the OU structure in Figure 13-5. The first three letters are the airport code for the nearest major airport, followed by a two-digit number identifying the specific location within a given city.

Environment Code
: This letter identifies whether the server is in a production (P), test (T), or development (D) environment.

Server Type Code
: This three- to four-character field generically identifies the server type. A list of server type codes used by Tailspin Toys is documented in Table 13-1.



We have successfully used naming conventions very similar to the one documented here on numerous occasions. Do not define a server type code for every single permutation of server you can imagine. If you don't have a need to differentiate different types of servers on the network by name, then stick with a generic type code.

We often use the application server type code when there is not a specific type code defined for a new server that is being introduced.

Physical/Virtual/Cluster Indicator
: This letter identifies whether the server is running on physical hardware (P) or in a virtualized environment (V), or if the name represents a cluster virtual name (C).

Index
: This is a numeric field, beginning with 01, used to disambiguate multiple servers in a given site of the same type.

Table 13-1. Server type codes

Server type code	Description
ADC	Domain controller
ADM	Management tools server
APP	Application server
CLN	Cluster node
DBS	Database server
RDS	Remote Desktop server
SRV	File and print server
VMH	Virtualization host
XSV	Exchange server

Step 5: Plan for users and groups

Given the relatively small size of Tailspin Toys, a simple user naming convention comprised of the first initial of the user's first name followed by the user's last name is devised. For example, George Washington's username would be *gwashington*. In the event of a conflict, a two-digit sequential number is appended to the end of the username to guarantee its uniqueness.

Since all groups will be centrally managed by the IT Department, names are assigned on an ad hoc basis that best corresponds to the purpose of the group. A policy is in place that requires the purpose of the group as well as two functional owners for the group to be listed in the `notes` attribute of each group. The group's functional owners are responsible for approving additions of users to groups as well as reporting on what the group is used for.

Contoso College

Contoso College is a higher education organization with approximately 3,500 employees, 30,000 students, and 5,000 computers. All users and computers are located at a single campus. Administration of IT resources is decentralized, with individual departments maintaining large-scale autonomy over the computers and resources in their areas. In an effort to provide a centralized, sustainable, and distributed directory service, central IT is implementing an Active Directory forest that supports automatically managed user accounts and groups as well as the ability for delegated administrators to receive OUs whose resources they can manage as they wish.

Step 1: Decide on the number of domains

Given Contoso's plans to include students in the central Active Directory forest, there is a great deal of debate surrounding whether or not student accounts should be placed in a separate child domain or even in a separate forest. A separate student account forest

is quickly ruled out, given the need for a central directory for applications to interact with, as well as the need for straightforward management. After reviewing the facts, including the reality that child domains do not create security boundaries, Contoso College decides to move forward with a single-domain forest that will contain all of the user accounts.

Step 2: Design and name the tree structure

Contoso already has an existing DNS infrastructure running on Unix BIND servers. Since the Windows administration team will manage Active Directory, and the Unix BIND environment is not configured to support dynamic updates, Active Directory will be deployed in a subzone of Contoso's root domain (*contoso-college.edu*). The subzone will be delegated to the Active Directory domain controllers and hosted in an Active Directory–integrated DNS zone. In order to provide a generic name for the domain, the DNS name of *net.contoso-college.edu* is chosen. The NetBIOS name of *NET* will be used.

Step 3: Design the hierarchy of organizational units

Contoso needs to satisfy a number of sometimes competing demands while planning the OU structure. The first is for a domain that is manageable by central IT resources while also being flexible enough to meet the needs of delegated administrators in various departments at the college. In order to meet these needs, Contoso develops the top-level OU structure shown in Figure 13-9.

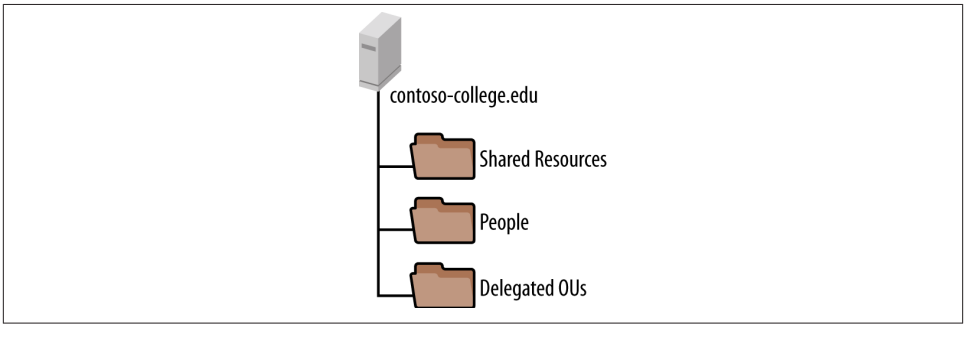


Figure 13-9. Contoso College top-level OU structure

Here is a description of each top-level OU in Figure 13-9:

Shared Resources
: This OU contains objects that relate to the support of the forest and the network as a whole. These resources include administrative accounts, centrally managed groups, and centrally managed servers (excluding domain controllers).

People
: Since user objects will be managed by an automated identity management solution, it makes the most sense for those objects to be in a centrally managed OU. Contoso decides not to split user accounts by user type (e.g., employee or student) or location, since many users at the college have multiple roles and sometimes multiple jobs. It is not uncommon for an employee to be taking a class and thus be considered a student as well, for example.

Delegated OUs
: Under this OU, each OU that is requested by a department administrator will be created. OUs are nested under the Delegated OUs OU in case there is a need to apply a group policy or security delegation to all of the objects without affecting shared resources, domain controllers, and so forth. We will look at this structure in more depth later in this section.

The structure for each delegated OU is a blend of a fixed structure that is common across the environment with the flexibility for delegated administrators to create OUs in certain locations. Figure 13-10 shows the fixed structure that is created under each delegated OU.

Within each delegated OU, there are a number of child OUs:

Groups
: Any group object that delegated administrators create is placed in this OU. Additionally, a number of groups that are created with the OU structure and delegated permissions are placed in this OU. Delegated administrators can then manage rights to their OU structures by simply nesting users in the correct groups.

  For the Groups OU, a group called *<Department Name> - Group Managers* is created and delegated the ability to create, delete, and manage all of the groups in the department's Groups OU.

Users
: User accounts that correspond to students and employees are placed in the top-level People OU shown in Figure 13-9. Delegated administrators may create accounts for guests that need temporary access as well as service accounts for applications running on servers managed by the OU owner. Contoso's security policy states that the OU owner is responsible for managing the lifecycle of these accounts and is responsible for any actions taken on the network by these users.

  Two groups are created with permissions delegated over the OUs under the Users OU. A *<Department Name> - Guest Managers* group has the ability to create, delete, and manage user objects in the Guests OU. Likewise, the *<Department Name> - Service Account Managers* group has the ability to create, delete, and manage user accounts in the Service Accounts OU.

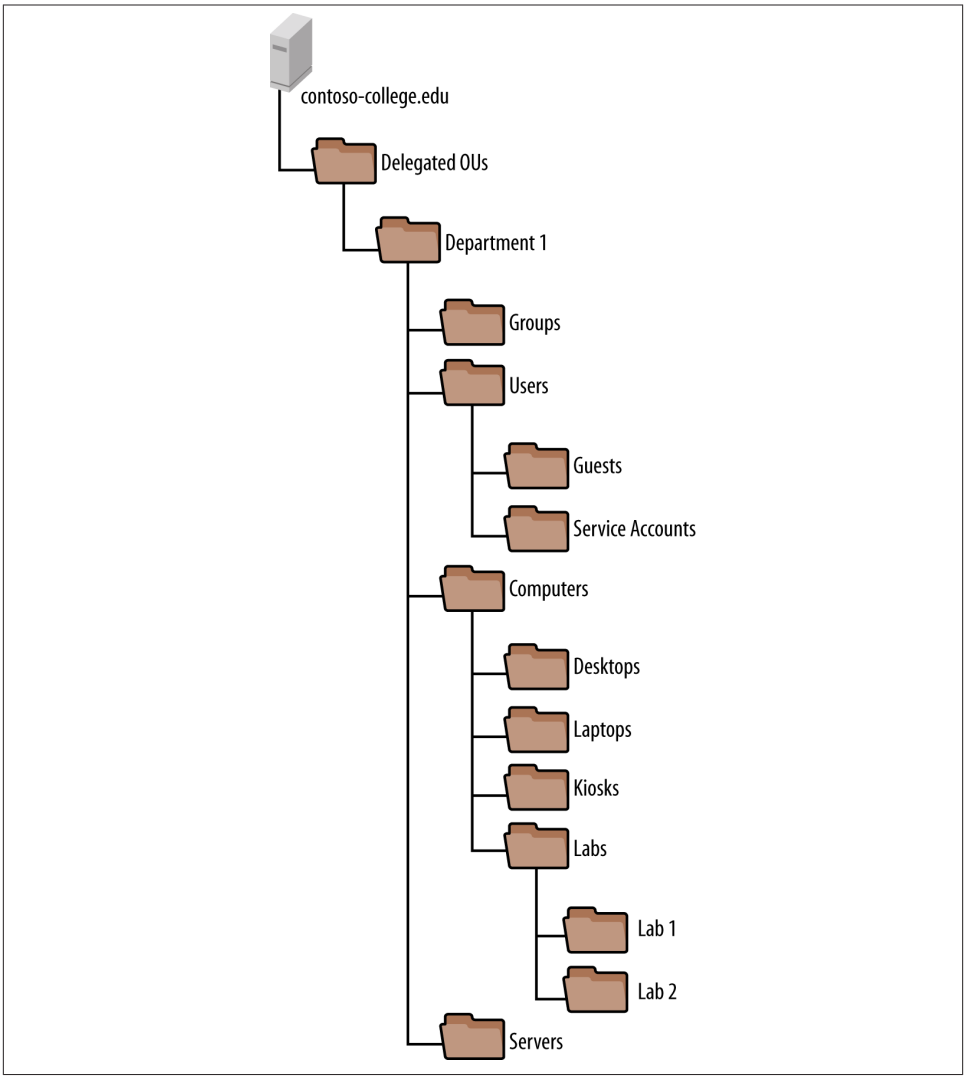


Figure 13-10. Contoso College department OU template

Computers
: The Computers OU has a number of child OUs. It is intended to hold all of the client computers managed by the OU owner. A group called <*Department Name*> - *Allow Domain Join* is created that is granted rights to join computers to the domain. This group also has rights on the default Computers container in the domain to reset computer account passwords and change computer account names.

The Desktops, Laptops, and Kiosks OUs are governed by a group called *<Department Name> - Computer Managers*. This group has the ability to manage computer objects in these OUs as well as move computers between OUs. Members of the group can also create child OUs under the Desktops, Laptops, and Kiosks OUs.

The Labs OU is specially designated for computer labs that typically contain student use computers. Computer labs often require special Group Policy configurations, so they are separated to ensure those policies are not applied to normal client computers. A group called *<Department Name> - Lab Computer Managers* has rights to manage computer objects under the Labs hierarchy as well as create child OUs.

Servers
: The Servers OU contains servers managed by the department. Contoso's security policy states that OU owners are responsible for the security of any servers that are managed by the department.

Step 4: Design the workstation and server naming conventions

Due to the distributed nature of Contoso College, there is no well-defined workstation or server naming convention in use across the organization. Departments are required to register a prefix of two to five characters with the central IT organization when they receive a delegated OU. All workstation and server names must be prefixed with this code. Computer accounts are automatically moved to the correct OU based on the prefix on the computer object's name.

Step 5: Plan for users and groups

Contoso College assigns a unique username to every user when they are entered into the college-wide enterprise resource planning (ERP) system. This username is reflected in Active Directory by the identity management system when the user's account is provisioned. If a user wants to change her username, she must request this change via the Human Resources or Student Affairs Department. If the change is approved, it will be entered into the ERP system and then automatically reflected in Active Directory.

In the case of manually created accounts such as guests, Contoso's security policy states that guest account usernames must be prefixed with *g-*. Additionally, these accounts must have an expiration date defined in Active Directory. Central IT automatically disables guest accounts that do not have an expiration date defined on a nightly basis.

Service accounts must be prefixed with *sa-* and the description field must explain the purpose of the account.

Groups may be named however the creator desires; however, the name must be prefixed with the two- to five-character code that is registered with IT when an OU is requested. This ensures that group names will not conflict.

Fabrikam

Fabrikam is a global multibillion-dollar organization that has more than 250,000 employees and computers located at over 100 sites around the world. The business has its global headquarters in Chicago. There are four other major sites that link to the HQ and to which the smaller branch offices link: Asia-Pacific, North America, South America, and Europe. The small sites link to the hubs via links ranging from 64 kbps to T1 speeds; the hubs themselves connect to the HQ via a variety of higher-speed links. Some of the hubs are also interconnected.

Fabrikam's management structure is regionalized, with each geographical unit running itself as an independent business as part of the global organization. The top level of the management structure is located at headquarters, which sits above the four hubs. Even though Chicago could be considered within the North America region, the organization is not structured that way. In fact, IT personnel in Chicago oversee the hubs in terms of selecting the administrators and determinig how the network is to be structured. Corporate IT policy dictates that branches that have more than 500 people have their own local administrator, backup support, and helpdesk staff. Branches with fewer than 500 people have to be managed by the administrators of the hub to which they connect (see Figure 13-11).

Other considerations include the following:

- Fabrikam recently acquired Trey Research, a Canadian company that has a strong brand name that Fabrikam would like to maintain. Trey Research is solely based in a new branch in Canada.
- The links between the South American branch offices and the hub are very unreliable.
- The Asia-Pacific hub site link to Europe is often severely congested.

Step 1: Decide on number of domains

Traditionally, if Fabrikam were designing its Active Directory installation in the Windows 2000 time frame, it would likely have opted to deploy an empty root domain with a series of child domains organized by region (e.g., domains for North America, Europe, South America, and the Asia-Pacific region). Considering the size of the directory and the network connectivity challenges that Fabrikam faces in South America and Asia, it is not out of the question that segregating replication by deploying child domains in some regions may be an appropriate and necessary solution.

Considering the relative cost and complexity of deploying a multidomain forest, as well as the lack of complete performance data and impact analysis around the network, Fabrikam decides to move forward with a single-domain forest. When considering the impact of replication, Fabrikam recognized that the benefits of a multidomain forest

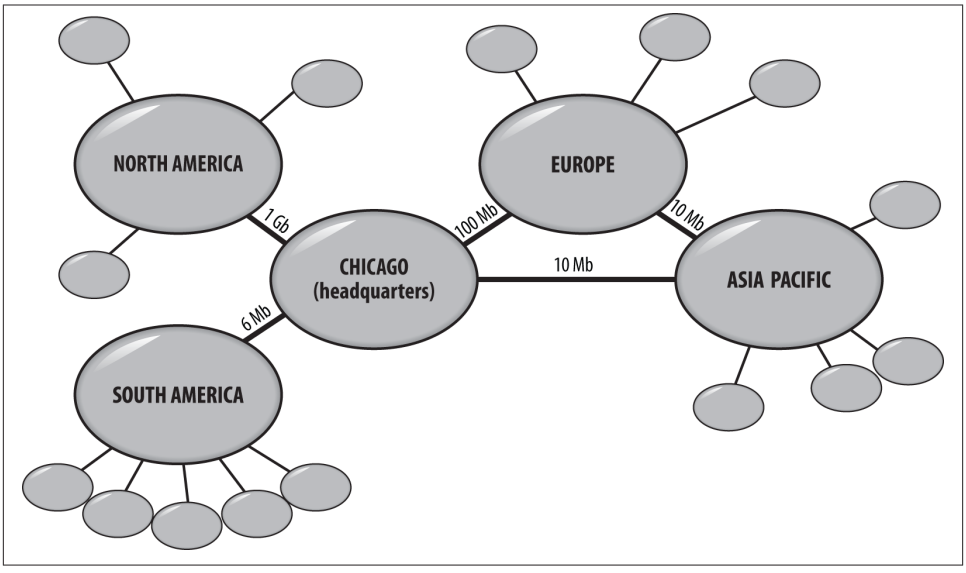


Figure 13-11. Fabrikam network topology

would be somewhat limited in the branch offices with domain controllers, as they would still require local Global Catalog presences. In the event that it becomes apparent that performance in South America or Asia is adversely impacted by using a single global domain, Fabrikam will deploy a child domain in that region and migrate users, computers, and other resources into the child domain.

Trey Research is an independent subsidiary of Fabrikam that Fabrikam intends to maintain as a separate company. Trey Research has contracts with Fabrikam competitors, and management feels there is a potential that Trey Research may be divested in the future. As a result, Trey Research will continue to maintain a separate forest with a two-way trust to the Fabrikam forest.

Step 2: Design and name the tree structure

Fabrikam has an existing DNS infrastructure, and the cost of moving to a split-brain DNS model for the *fabrikam.com* zone is prohibitive. Currently, Fabrikam owns the *fabrikam.net* domain name, but the namespace is not being used for anything. Consequently, Fabrikam decides to deploy its forest root domain as *fabrikam.net* with a NetBIOS name of *FABRIKAM*.

Once the forest is operational, a two-way forest trust is established with the *treyresearch.com* domain.

Step 3: Design the hierarchy of organizational units

Given Fabrikam's semi-decentralized model of IT administration, a good number of permissions will need to be delegated to technicians at branch offices that manage IT resources locally. All of the IT personnel across the company are ultimately accountable to the headquarters IT staff in Chicago, so control will not be delegated such that local personnel have *carte blanche* access to their respective locations.

Instead, a standardized structure will be put in place for each location. Local personnel will have access to perform tasks relevant to their jobs, and IT personnel at the headquarters office will have access across all locations in the company.

Figure 13-12 shows the top-level structure for the Fabrikam domain.

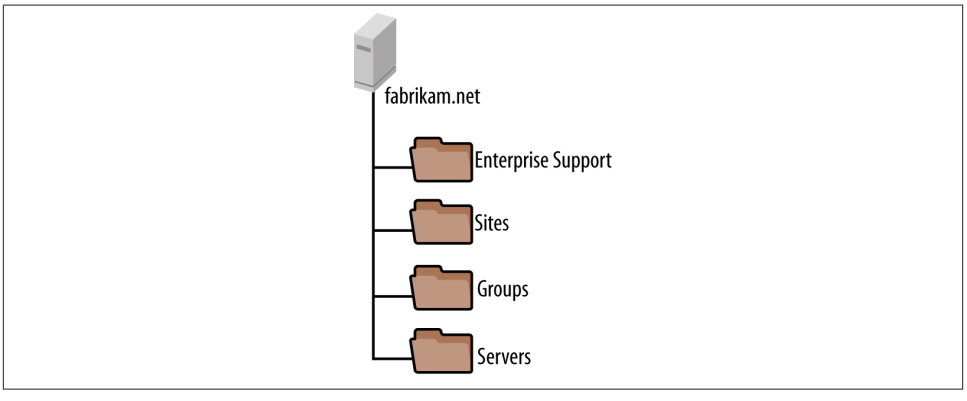


Figure 13-12. Fabrikam top-level OU structure

Here is a description of each top-level OU:

Enterprise Support
: This OU contains objects that relate to the support of the forest and the network as a whole. These resources primarily include administrative accounts and service accounts.

Sites
: This OU is the parent for OUs that represent each Fabrikam office. We'll look at the structure for this OU shortly.

Groups
: Fabrikam elects to manage all groups through a central, web-based group management tool. Consequently, groups are all placed in a single OU that has no delegated access.

Servers

Fabrikam's IT security policy states that all servers of a specific function must have the same policy and settings applied to them. Servers are organized into child OUs by the type of service or application running on the server so that a group policy can be applied to all servers of that type.

The Sites OU is shown in Figure 13-13. Each location-specific OU is organized by region, country, and then site. This allows administrators to delegate control at each level of the hierarchy (for example, to grant helpdesk analysts in Singapore the ability to reset user passwords for all users in Asia), as well as to apply group policies at any level of the hierarchy.

Before you start building all of this in a GUI like Active Directory Users and Computers, pull the *Active Directory Cookbook* by Laura E. Hunter and Robbie Allen (O'Reilly) off the bookshelf and put together a script to build the OU structures and permissions. Since the structures will all be identical, the best mechanism to maintain consistency over time is to script the process. Administrators don't want to be data-entry clerks, so building a structure like this is a perfect task for a script.

Step 4: Design the workstation and server naming conventions

Fabrikam is using a desktop management solution to keep track of all its desktops and laptops. Machines also move around the company fairly frequently, and employees often do not stay employed with the organization for long periods of time. Consequently, Fabrikam doesn't want to encode information about the location of the computer or the employee using the machine in the name. At the same time, Fabrikam needs to ensure computer names are unique on the network and have a way of correlating a given machine to its record in the desktop management database.

To achieve this, Fabrikam decides to name each machine based on its manufacturer's serial number. Since Fabrikam uses multiple suppliers for desktops and laptops, it decides to prefix the machine's serial number with a two-character code for the machine's manufacturer, as shown in Table 13-2. This naming convention is easy for technicians to implement, never requires a change in the machine's name, and guarantees correlation of the machine name with the asset management database.

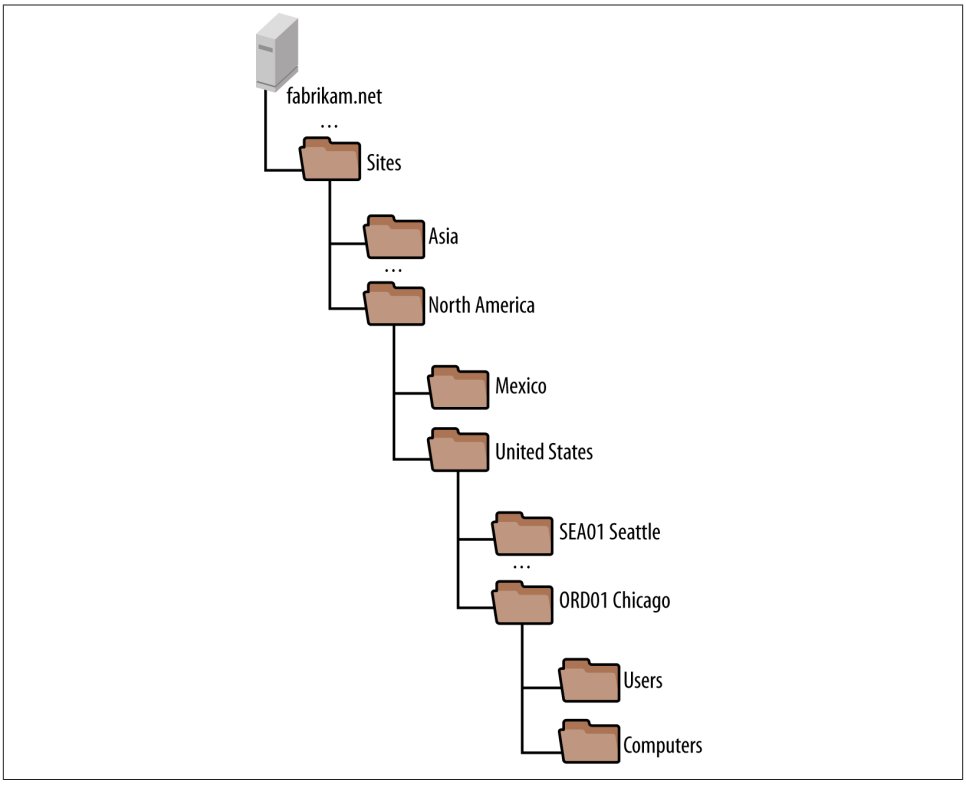


Figure 13-13. Fabrikam site-level OU structure

Table 13-2. Computer naming prefixes

Code	Manufacturer
AP	Apple
DL	Dell
HP	Hewlett-Packard (HP)
IB	IBM
LN	Lenovo

For servers, Fabrikam maintains information about each server and what its role is in a configuration management database (CMDB). Since all of this information is tracked in the CMDB, Fabrikam decides to name servers based on a location code for the site hosting the server followed by a sequential four-digit number that uniquely identifies the server at that location. For example, a server at the SEA01 site might be named *SEA01-0096*.

Step 5: Plan for users and groups

User accounts will be provisioned automatically from the central human resources (HR) database maintained at the Chicago headquarters. Due to the sheer number of employees and the potential for duplicates, the provisioning system constructs a unique username for each user that is comprised of portions of the user's first name, middle initial, and last name.

Groups are named arbitrarily by the user that creates the group through the central group management system.

Recognizing Nirvana's Problems

Arguably, there are a number of "best" ways to design, depending on whom you talk to. We propose an iterative approach with Active Directory, and this is probably going to happen anyway due to the nature of the many competing factors that come into play. On your first pass through this chapter, you'll get a draft design in hand for the namespace. In Chapter 14, you'll get a draft site and replication design. Then you'll come up against the issue that your namespace design may need changing based on the new draft site and replication design, specifically with regard to the issues of domain replication and server placement that we have just covered. After you've revised the namespace design, you can sit down and look at the GPO design (referring to Chapter 11 and Chapter 15) in a broad sense, as this will have an impact on the OU structure that you have previously drafted in your namespace design.

While this is the way to design, you will come up against parts of your organization that do not fit in with the design that you're making. The point is to realize that your job is to identify a *very good* solution for your organization and then decide how to adapt that solution to the real world that your company inhabits. One domain may be ideal but may not be practicable in terms of cost or human resources. You have to go through stages of modifying the design to arrive at a compromise solution that you're happy with.

Finally, we want to stress that the case studies in this book are only examples! There are multiple ways each of these cases could have been designed. We chose designs that we felt would work given the limited information available about each sample organization as well as the amount of space available to us in this book.

Do not assume that the designs you saw in this section are indications of how you should design your directory. We think they are excellent starting points, but you need to take your requirements and develop a design that fits best for your organization.

Summary

In this chapter, we presented five steps toward effective namespace design:

- Decide on the number of domains.
- Design and name the tree structure.
- Design the hierarchy of organizational units.
- Design the workstation and server naming conventions.
- Plan for users and groups.

Following these 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.

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

The rest of the design still needs to be completed, though. 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 14

Creating a Site Topology

As we mentioned in Chapter 6, there are two aspects to replication:

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

We covered the former in Chapter 6, 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.

Intrasite and Intersite Topologies

Two distinct types of replication connections 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 connections between intrasite DCs. The KCC will create intersite connections automatically for you as well, but only when an administrator has specified that two sites should be connected via a *site link*. Every aspect of the KCC and the connection objects that are created is configurable, so you can control what has been automatically created and what will be automatically created via manipulation of the various options.

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 connections created by the KCC uses a notification process to announce that changes have occurred—each domain controller is responsible for notifying its replication partners of changes. If no changes occur at all within a six-hour

period, the replication process is kicked off automatically just to make sure that nothing was missed. Intersite replication, on the other hand, does not use a notification process by default. Instead, intersite replication relies on a schedule to transfer updates, using compression to reduce the total traffic size.

The KCC

Domain controllers within sites have connections created between them by the KCC. These connections use a random GUID as the unique identifier and are represented in Active Directory as *connection objects*. When domain controllers between sites must be connected, the Intersite Topology Generator (ISTG) automatically creates connection objects in Active Directory between these domain controllers. Within each site, an ISTG is designated to generate the intersite topology for that particular site via the KCC process.

The ISTG depends upon the presence of site links in your Active Directory network in order to create the connections between domain controllers in different sites. Site links are an administratively defined construct that generally represents the network topology on top of which an Active Directory deployment exists. We will cover site links in detail later in this chapter. 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 manually on demand if required (see the sidebar "Triggering the KCC and ISTG"). If we create two domain controllers called Server A and Server B in a new domain, the KCC will run on each server to create links. Each KCC is tasked with creating a link to define incoming replication only. The KCC on Server A will define an incoming link from Server B, and Server B's KCC will define an incoming link from Server A. The KCC creates only one incoming link per replication partner, so Server A will never have two autogenerated incoming links from Server B, for example.

Triggering the KCC and ISTG

The KCC and the ISTG can both be forced to run on demand with the *repadmin* command-line tool.

To force the KCC to run on the local domain controller, run this command:

```
repadmin /kcc
```

To force the KCC to run on the *coho-chi-adc01.cohovines.com* remote domain controller, run this command:

```
repadmin /kcc coho-chi-adc01.cohovines.com
```

To force the ISTG to run on the local domain controller, run this command:

```
repadmin /istg
```

To force the ISTG to run on the *coho-chi-adc01.cohovines.com* remote domain controller, run this command:

```
repadmin /istg coho-chi-adc01.cohovines.com
```

The KCC does not create one topology for all naming contexts (NCs) or one topology per NC. The Configuration and Schema NCs share one replication topology, so the KCC creates a topology for these two together. The KCC also creates another topology on a per-domain/naming context basis. Because the Schema and Configuration NCs are forest-wide in scope, the KCC needs to replicate changes to these items across site links. The KCC needs to maintain a forest-wide topology spanning all domains for these two NCs together. However, unless a domain is set up to span multiple sites, the topology for a particular domain will be made up of only intrasite connections. If the domain does span sites, the KCC needs to create a replication topology across those sites.

Automatic Intrasite Topology Generation by the KCC

For each naming context, the KCC builds a bidirectional ring of replication connections between the domain controllers in a site. Upstream and downstream connections are created between partners around a ring, but the KCC can create additional connections across the ring as well. These connections are created on the basis of a single rule: that no DC in the site can be more than three replication hops from any other DC in the same site. The KCC does this to maintain a guaranteed theoretical maximum convergence time within a site for an NC. Generally speaking, inside of a site, this convergence time is approximately one minute. If you want to modify the speed of convergence, refer to the sidebar "Modifying Replication Convergence Intervals." For example, as you add new DCs for a domain to a site, they will just get inserted into the ring. Due to the three-hop rule, when you put in your eighth DC, the KCC must add at least one cross-ring connection. Let's take a look at this process in more detail.

Modifying Replication Convergence Intervals

If you want to change the default replication intervals within a site, you can do so by modifying the registry on specific domain controllers, or by setting the values in Active Directory so that all domain controllers hosting a given naming context are affected.

If you want to set these values on specific domain controllers, create or modify the following registry values (`REG_DWORD`s) in the `HKLM\SYSTEM\CurrentControlSet\Services\NTDS\Parameters` key:

- To change how long to wait before notifying the first change-notification partner of an originating write, set `Replicator notify pause after modify (secs)`. The default value is 15 seconds.
- To change how long to wait after notifying each additional change-notification partner of an originating write, set `Replicator notify pause between DSAs (secs)`. The default value is 3 seconds.

If you want to modify the timings on a global basis for a given naming context, set the following attributes on the cross-reference objects for the specific NCs:

- `msDS-Replication-Notify-First-DSA-Delay`
- `msDS-Replication-Notify-Subsequent-DSA-Delay`

This will impact all domain controllers replicating the NC, so it isn't as targeted as the registry settings.

Two servers

In the case of two domain controllers, we start off with one DC, Server A. When Server B is promoted as the second DC for the domain, the *dcpromo* process uses Server A as its source for Active Directory information for the Domain, Schema, and Configuration naming contexts on Server B. During the promotion process, the Configuration container is replicated from Server A to Server B, and the KCC on Server B creates the relevant incoming connection object representing Server A. Server B then informs Server A that it exists, and Server A correspondingly creates the incoming connection object representing Server B. Replication now occurs for all NCs using the connection objects. While replication occurs separately for each NC, the same connection objects are used for all three at this moment.

Three servers

The *dcpromo* process is later started on Server C. Server C then uses a DNS lookup and picks one of the existing DCs to use as a promotion partner. For now we'll say that it picks Server B. During the promotion process, the Configuration container is replicated from Server B to Server C, and Server C creates the relevant incoming connection object representing Server B. Server C then informs Server B that it exists, and Server B correspondingly creates the incoming connection object representing Server C. Replication now occurs for all NCs using the connection objects.

At present, you have two-way links between Server A and Server B and between Server B and Server C. There are no links between Server A and Server C, but the KCC must create a ring topology for replication purposes. So, as soon as Server B does a full replication to Server C, Server C knows about Server A from the Configuration NC. Server

C's KCC then creates an incoming connection object for Server A. Server A now finds out about Server C in one of two ways:

- Server A requests updates from Server B and identifies a new DC.
- Server C requests changes from Server A, and this allows Server A to identify the new DC.

Server A now creates an incoming connection object for Server C. This completes the Server A to Server B to Server C to Server A loop.

Four servers

Server D comes along, and the promotion process starts. It picks Server C to connect to. Server D ends up creating the incoming connection object for Server C. Server C also creates the incoming connection object for Server D. You now have the loop from the previous section, plus a two-way connection from Server C to Server D. See Figure 14-1 for this topology.

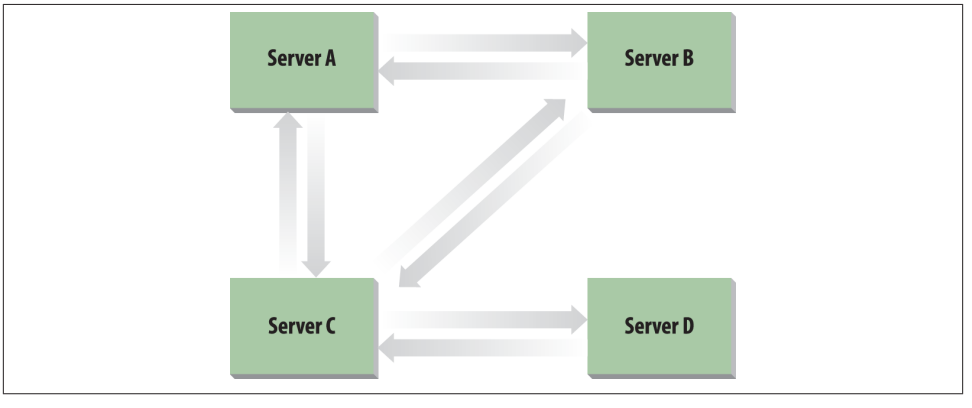


Figure 14-1. Adding a fourth DC to a site

Server D's KCC now uses the newly replicated data from Server C to go through the existing topology. It knows that it has to continue the ring topology, and as it is already linked to Server C, Server D has to create an incoming connection object for one of Server C's partners. It chooses Server B in this case. So, Server D's KCC creates an incoming connection object for Server B. Server D then requests changes from Server B. The rest of the process can happen in a number of ways, so we'll just play out one scenario.

Server B now knows about Server D. Server B's KCC runs and realizes that it doesn't need the link to Server C, so it deletes that connection and creates a new one directly to Server D itself. Finally, as replication takes place around the ring along the existing links,

Server C notes that it has a now-defunct incoming link from Server B and removes it. You now have a simple ring, as depicted in Figure 14-2.

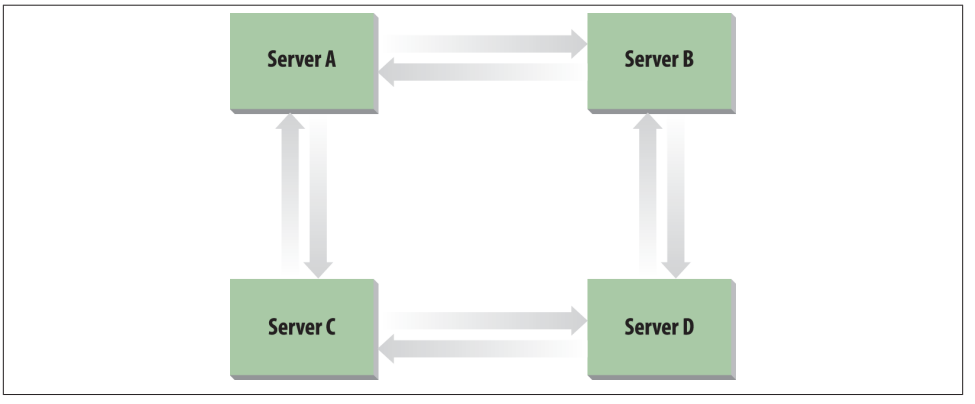


Figure 14-2. Ring of four DCs

Eight servers

Once you hit eight servers connected together, you need more links across the ring if you are to maintain the three-hop rule. If you look at Figure 14-3, you will see this demonstrated. If the cross-ring links did not exist, some servers would be four hops away from one another. The KCC figures out which servers it wishes to link by allowing the last server to enter the ring to make the initial choice. Thus, if Server H is the newest server in the ring, it knows that Server D is four hops away and makes a connection to it. When Server D's KCC receives the new data that Server H has linked to it, it reciprocates and creates a link to Server H.

However, this doesn't completely solve the problem. Consider Server B and Server F: they're still four hops away from each other. Now the KCC creates a link between these pairs to maintain the three-hop rule.

Now what?

We've now gone through the mechanism that the KCC uses for intrasite link generation between DCs. However, that's not the whole story. Remember that Active Directory can have multiple domains per site, so what happens if we add *contoso.com* (a new domain in the same forest) to the same site, or even *europe.cohovines.com* (a new child domain)? The answer is the same for both, and it is based on NCs:

- The Schema and Configuration NCs replicate across the enterprise, and they share a replication topology. Although they replicate separately, it is along the same links.

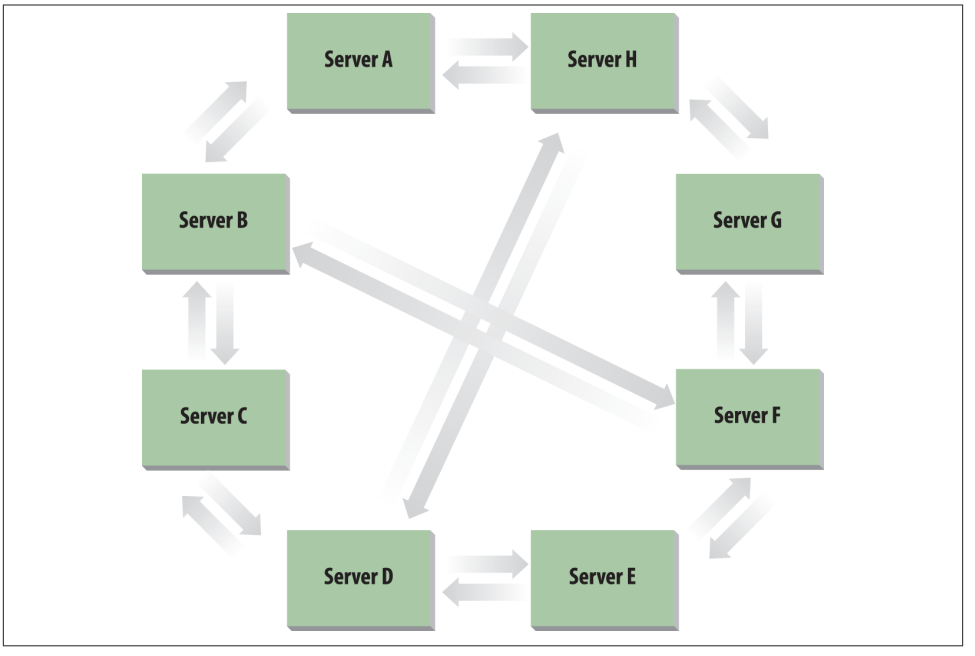


Figure 14-3. Ring of eight DCs

- Each domain replicates only domain-wide, so the domain topologies for both domains stay in the same ring formation that they had previously.

Once the two domains integrate, the KCC-generated topologies for *cohovines.com* and the other domain stay the same. However, the KCC-generated Configuration/Schema replication topology that exists separately on both domains will form itself into its own ring, encompassing both domains according to standard KCC rules.

To summarize, when you have multiple domains in a site, each domain has its own KCC-generated topology connecting its DCs, but all the DCs in the site—no matter what domain they come from—connect in a separate topology representing Schema/Configuration replication.

Site Links: The Basic Building Blocks of Intersite Topologies

Having sites is all well and good, but you need to be able to connect them if you are ever going to replicate any data. An intersite connection of this type is known as a *site link*. Site links are created manually by the administrator and are used to indicate that it is possible for two or more sites to replicate with each other. Site links can connect more than two sites if the underlying physical network already connects multiple sites

together; however, it is often easier to visualize and control replication connections if you limit yourself to two sites per site link.

Sites do not have to be physically connected by a network for replication to occur. Replication can occur via multiple links between any two hosts from separate sites. However, for Active Directory to be able to understand that replication should be occurring between these two sites, you have to create a site link between them.

We've mentioned that site links have a cost (see Chapter 6), but that's not their only property. In fact, site links have four important properties:

Name
: An identifying name for the site link.

Cost
: An integer weighting for the site link that indicates the preference of the link relative to the other links that exist. Lower costs are more preferable; higher costs are less preferable.

Schedule
: The times that are available for replication to begin. Replication will not begin outside of the time windows specified in a schedule.

Transports
: The protocols that are used for replication along this link.

Cost

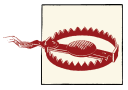
As each link has a cost, it is possible to calculate the total cost of traveling over any one route by adding up all the costs of the individual routes. If multiple routes exist between two sites, the KCC will automatically identify the lowest-cost route and use that for replication. If only a single route exists between any two sites, then the site link costs are irrelevant.

Schedule

The schedule on a link represents the time period when replication is allowed to be initiated across that link. Domain Controllers also maintain times that they are allowed to replicate. Obviously, if two DCs and a link do not have times that coincide, no replication will ever be possible.

Between the scheduled start and stop times for replication on a site link, the server is available to open so-called *windows* for replication to occur. As soon as any server that replicates through that link becomes available for replication, a replication window is opened between the site link and that server. As soon as two servers that need to replicate with each other have windows that coincide, replication can occur. Once a server

becomes unavailable for replication, the window is removed for that server. Once the site link becomes unavailable, all windows close.



Schedules only define when replication can begin. Once replication has begun, it will not stop until it completes, regardless of the schedule defined on the site link.

Transport

Site links can currently replicate using two transport mechanisms:

- Directory Service Remote Procedure Call (DS-RPC)
- Inter-Site Messaging Simple Mail Transport Protocol (ISM-SMTP)

A site link using DS-RPC means that servers wishing to replicate using that site link can make direct synchronous connections using RPC across the link. As the transport protocol is synchronous, the replication across the connection is conducted and negotiated in real time between two partners. This is the normal sort of connection for a real-time link. In some scenarios, certain sites may have unpredictable availability or they may have a very unreliable or highly latent WAN connection. In these scenarios, SMTP replication can be appropriate.

The *SMTP connector*, as a site link using the ISM-SMTP transport is called, allows partner domain controllers to encrypt and email their updates to each other. In this scenario, Active Directory assumes that you already have an underlying SMTP-based connection mechanism between these two sites. If you don't, you'll have to set one up for this to work. If a connection is in place, the SMTP connector assumes that the existing underlying mail routing structure will sort out how mail is transferred. To that end, a site link using the SMTP connector ignores the scheduling tab, as it will send and receive updates automatically via the underlying system whenever the email system sends and receives them itself.

SMTP connector messages are encrypted using certificates, so to encrypt the messages you need to install special certificates on the domain controllers that will participate in SMTP replication. You can automatically request and install these certificates if you have an Enterprise Certification Authority (CA) running on Windows in your organization. Otherwise, reference Microsoft Knowledge Base article 321051 for instructions on requesting and installing certificates from a third-party CA.

The SMTP replication protocol cannot be used for Domain NC replication. It can, however, be used to replicate Global Catalog, schema, and configuration information. This means that multisite domains with slow links will be required to use RPC for domain replication.

When the ISTG becomes involved

In order to generate the intersite replication topology, the ISTG actively uses site link costs to identify which routes it should be using for replication purposes. If a stable series of site links exists in an organization, and a new route is added with a lower cost, the ISTG will switch over to use the new link where appropriate and delete the old connection objects. The network of connections that the KCC creates is known as a *minimum-cost spanning tree*.

Site Link Bridges: The Second Building Blocks of Intersite Topologies

Site link bridges are only necessary in scenarios where you have disabled the "Bridge all site links" option in Active Directory. We discussed an example of when a site link bridge may be necessary in Chapter 6. To summarize, site link bridges are necessary when you wish to tell the KCC that it can create a direct replication connection between two domain controllers that are not directly connected according to the topology information in Active Directory.

Consider the sample topology in Figure 14-4, and assume for the purposes of this discussion that "Bridge all site links" is disabled in the sample topology and that the underlying network allows communication between Chicago and New York. Given this information, Chicago and New York cannot replicate directly with one another. Instead, Chicago must replicate with Denver, and New York must replicate with Denver as well. In the event that the Denver domain controller (*DC02*) goes down, there will be no replication between Chicago and New York. If you define a site link bridge that includes the Chicago-Denver and New York-Denver site links, however, Active Directory will be able to create a replication connection directly between Chicago and New York in the event that the Denver domain controller becomes unavailable.

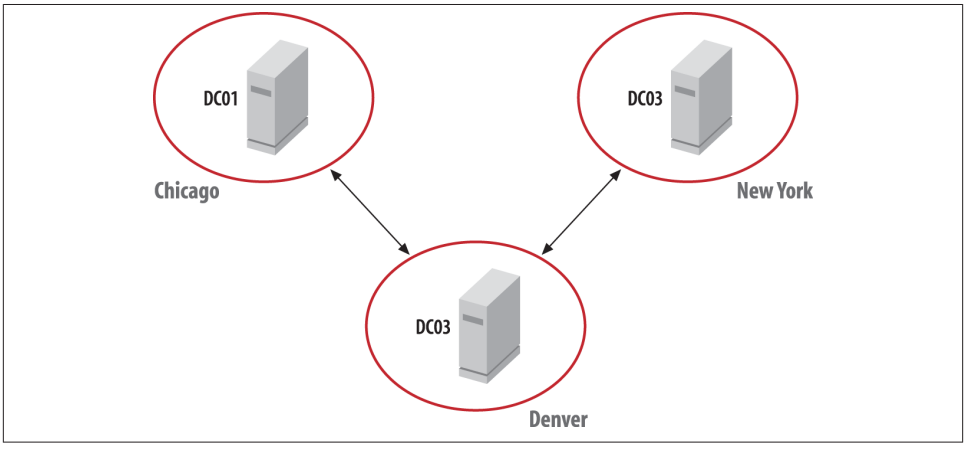


Figure 14-4. Site link bridging scenario

Now that you've seen how site links and site link bridges work, let's look at how to design your sites and their replication links.

Designing Sites and Links for Replication

There is only one really important point that 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, finding the nearest DFS share mount point, locating System Center Configuration Manager distribution points, or routing email in an Exchange organization. This section looks at the steps involved in designing your sites and links for replication.

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.

Additionally, you should write down all the subnets that correspond to the sites you have noted.

Step 2: Plan the Domain Controller Locations

Planning for placement of domain controllers is fairly easy, but determining the number of DCs to use is a different matter entirely.

Where to put domain controllers

Each workstation in a domain exists in a single site that it knows about. When a user tries to log onto the domain at a given 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 by way of the site topology and, by a process of negotiation with a DC in that site, either authenticates with that DC or is redirected to a better domain controller.

This consideration governs the placement of DCs. You should place one DC for authentication purposes per domain in all sites that meet any of the following criteria:

- The site has links that are not fast enough for logon purposes to a particular domain.
- The site has links that may be fast enough for logon, but you do not wish to authenticate across them for a particular domain.

- An application that requires fast responses from a domain controller (such as Exchange) exists in the site.
- You have overarching business requirements to ensure that authentication and directory information in that site will always be available, regardless of the WAN link status.

The first and second points also need to be considered in light of the number of users and workstations at the sites. If there are enough workstations at the site to generate logon traffic that will utilize a large percentage of the available WAN bandwidth, then you will probably be forced to place a local domain controller at the site.

How many domain controllers to have

Deciding how many DCs to place at a site is never easy. If you have a Windows server that's already serving 500 heavy users and is close to its load limit, could it authenticate 100 additional users quickly enough at the same time? Powerful servers can authenticate hundreds or thousands of users simultaneously, but even these servers will balk if they are already heavily loaded.

There are some things that are tough to accurately assess, such as the impact of LDAP-based applications using the DCs. Often you do not know the specifics of how any given application uses the directory, so you end up taking a "wait and see" stance in terms of how many DCs will be needed and hoping that the usage levels aren't so high that users will notice any performance problems.

Exchange is an example of an application that ships with best practice guidelines for how many Global Catalog servers are required. Exchange defines these best practices in terms of a ratio of Exchange CPU cores to Global Catalog CPU cores.

The only way to definitively decide on how many domain controllers are required in a site is often to try the best practice and recommended guidelines, and then adjust the actual count to suit your reality as patterns become clear. That way, you should be able to judge for yourself how many DCs you may need for authentication and other purposes.

It is important that you define up front when planning your domain controller deployment what criteria you will use to justify an additional domain controller in a site. This usually depends on performance metrics such as average CPU utilization or disk utilization.

Placing a domain controller in more than one site

Any domain controller that you promote will belong to one site only. However, there can be instances in which you may want to configure a domain controller to cover multiple sites. For example, you might want to make sure that workstations from a number of sites all authenticate using one DC. Domain controllers perform a behavior by default called *automatic site coverage*, which registers the necessary DNS records for a domain controller to service multiple sites.

Automatic site coverage is important when you have sites without domain controllers, as clients in those sites will still need to authenticate and access Active Directory. In branch office scenarios, it is often ideal to modify this behavior so that domain controllers in branch offices don't cover other sites. You can find more information on configuring this behavior in Chapter 4 of the Windows 2003 Branch Office Deployment Guide. It is no longer necessary to configure this behavior for Windows Server 2008 R2 and newer domain controllers.

You can manually configure domain controllers to service multiple sites by modifying a registry value. To do this, edit the registry on the server that will be covering multiple sites and add a `REG_MULTI_SZ` value called `SiteCoverage` to the `HKLM\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters` key. Add the names of the sites you want the server to cover to this value.

Generally speaking, you should plan to rely on automatic site coverage; however, this manual functionality exists in case you need to make an exception.

Step 3: 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 topologies, sites often are represented by ovals.

Remember that a site is loosely defined as a well-connected set of subnets ("well-connected" tends to mean about 10 Mbps LAN speed). A site does not have to have a

domain controller 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, though, because DCs in both physical locations will maintain the intrasite replication ring topology. If you had created two sites and a site link, only the two bridgehead servers would replicate with each other.

To summarize, we would suggest that, by default, you create one site per physical location on your WAN, unless you do not feel a need to segregate replication between two physical locations.

One counterexample to this general recommendation is in the case of high-speed links between datacenters. Some organizations deploy multiple physically separate datacenters that are effectively one logical location on the network. In this case, it may make sense to represent those datacenters as a single site.

Step 4: Create Site Links

Now that you have all the sites down on paper, you need to think about the links. In this step, we identify how sites are connected and to what degree we need to replicate the physical topology in Active Directory. Additionally, we identify any site links that will require nonstandard replication schedules and the frequencies at which site links replicate.

The first thing to consider is how you will name your site links. We recommend that you limit yourself to two sites per site link and adopt a naming convention that makes it easy to identify the sites involved in a site link. One strategy that works well is to place the two sites involved in both the name and description attributes, reversing the order. For example, you might make the name of the site link *SiteA-SiteB* and the description *SiteB-SiteA*. This makes it easy to sort on either field in Active Directory Sites and Services and see which site links involve a given site.

When applying costs to your site links, you should make every effort to use a standardized system for the cost values you choose across the entire deployment. There are multiple ways to implement this costing model, and there isn't necessarily a "right" way to do it. Some organizations choose to assign link costs categorically, such that links between hub sites have a certain cost, links between hubs and spokes have another cost, and links between two spokes have a third cost. Other organizations choose to associate

the site link costs with the speed of the underlying WAN connection. Remember that either way, if there is only one path to a site, the cost is irrelevant.

For a discussion of using WAN speeds to populate site link cost values, see this website. This blog post also includes a spreadsheet that provides the costs of many common WAN link speeds.

The frequency is simply how often replication will occur over the site link. This value is specified in minutes and can range from 15 minutes to 1 week. You should choose this value based on the requirement for how up to date Active Directory data must be at either end of the site link, and also with respect to how much load replication will place on the underlying WAN link. If you do not have enough change volume to generate a significant amount of replication traffic, it will likely not be beneficial to you to make the frequency intervals very long.

The schedule is the final configurable option on site links. The site link schedule defines during what time windows replication is allowed to begin. "Begin" is an important word here, as once replication starts, it will not stop until it concludes, regardless of the site link schedule. Site link schedules are very helpful when working around WAN links that are saturated or unavailable during certain time periods.

Step 5: Create Site Link Bridges

If you elect to disable the "Bridge all site links" option, you will need to examine whether your topology requires site link bridges to help the ISTG generate your intersite topology. In many hub/spoke topologies, this step ends up being unnecessary, but you will need to examine yours to be sure.

Design Examples

Having considered the five steps, let's take another look at the three examples from the previous chapter and see what they will need in terms of their site topologies.

Tailspin Toys

Tailspin Toys is an organization that employs approximately 3,000 people across North America. There are approximately 4,000 client computers and several hundred servers spread across the Tailspin Toys locations. Office workers are located in a number of cities across the United States and Canada, as well as in home offices. Manufacturing operations take place in Mexico.

Step 1: Gather background data for your network

The Tailspin Toys network is relatively modern, with fast and reliable connectivity to all of the offices in the United States and Canada. Some of the manufacturing plants have less reliable connectivity. All manufacturing sites are considered business critical and must be able to operate independently for a period of time in the event of a network outage.

The organization's primary datacenter is located in San Francisco. There is a disaster recovery site located in Atlanta. The network topology is shown in Figure 14-5.

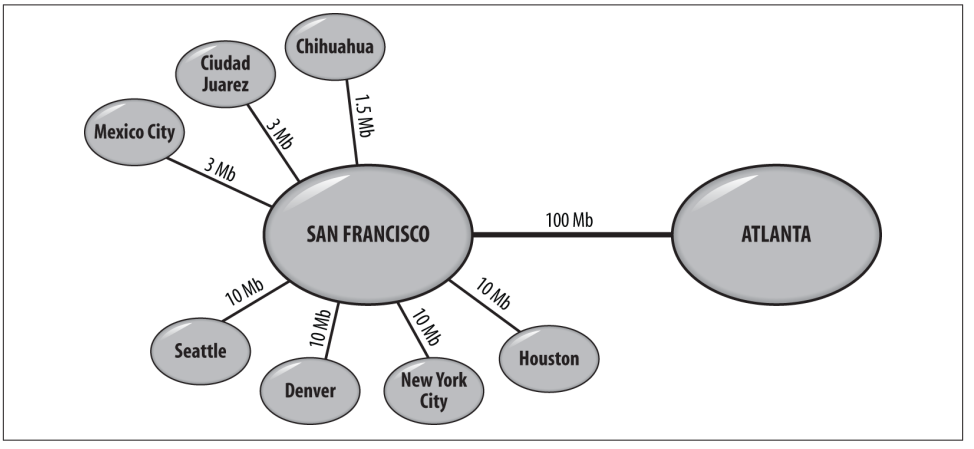


Figure 14-5. Tailspin Toys wide area network topology

Step 2: Plan the domain controller locations

Based on the network diagram in Figure 14-5 as well as historical network performance information, Tailspin Toys decides to locate writable domain controllers in its San Francisco and Atlanta datacenters to service clients in offices across the United States and applications hosted in the datacenters.

Given the critical nature of manufacturing operations, corporate policy, and WAN reliability, a read-only domain controller (RODC) is placed at each manufacturing site. The RODC is configured to cache passwords for all users, service accounts, and computers located at the site.

Step 3: Design the sites

Tailspin Toys currently does not use any applications that are both site-aware and deployed into sites that do not have a domain controller. As a result, site objects and corresponding subnets are created for each location hosting domain controllers. The San Francisco site is also configured with the subnet objects for each of the branch offices in the United States.

Step 4: Create site links

Figure 14-6 shows the site link topology for the Tailspin Toys forest. Table 14-1 shows the site link configuration. A single site link is provisioned for each connection to San Francisco. Since there is only one path to each site, site link costs are not relevant. Standardized cost values are assigned to each site link. All links between datacenters are assigned a cost of 100. All links between datacenters and spoke sites are assigned a cost of 200.

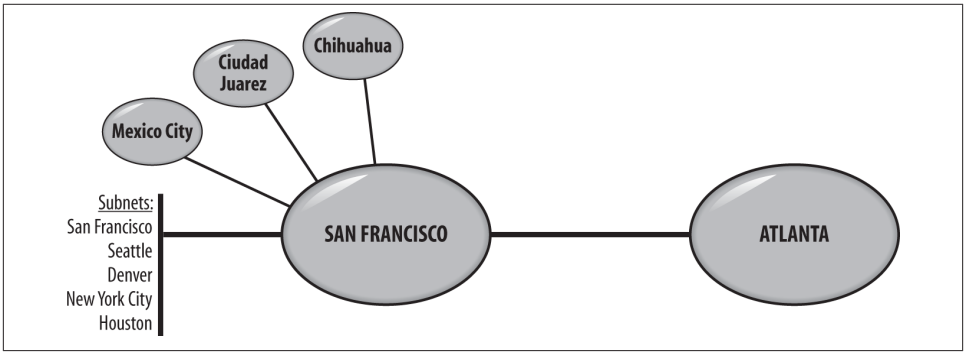


Figure 14-6. Tailspin Toys site topology

Table 14-1. Tailspin Toys site links

Sites included	Name	Description	Cost	Frequency	Schedule
San Francisco Atlanta	San Francisco-Atlanta	Atlanta-San Francisco	100	15 minutes (change notification enabled)	Always
San Francisco Mexico City	San Francisco-Mexico City	Mexico City-San Francisco	200	180 minutes	Always
San Francisco Ciudad Juárez	San Francisco- Ciudad Juárez	Ciudad Juárez -San Francisco	200	180 minutes	Always
San Francisco Chihuahua	San Francisco-Chihuahua	Chihuahua-San Francisco	200	180 minutes	Always

As you can see in Table 14-1, the high-speed link between the San Francisco and Atlanta datacenters also has change notification enabled. This ensures that domain controllers in both datacenters will always be virtually in sync. Since there is not a great need for data to be up to date on the RODCs in the manufacturing sites, replication is configured to occur every three hours (180 minutes) for those site links.

Contoso College

Contoso College is a higher education organization with approximately 3,500 employees, 30,000 students, and 5,000 computers. All users and computers are located at a single campus. There is a central datacenter managed by a central IT organization. In an effort to provide a centralized, sustainable, and distributed directory service, central IT is implementing an Active Directory forest that supports automatically managed user accounts and groups as well as the ability for delegated administrators to receive OUs whose resources they can manage as they wish.

Step 1: Gather background data for your network

The Contoso College campus is comprised of multiple buildings that are all connected to the central datacenter by high-speed fiber optic cables. The central datacenter is a physically secure facility with backup power.

Step 2: Plan the domain controller locations

Since the entire Contoso College campus is effectively a single well-connected local area network (LAN), writable domain controllers will be placed in the central datacenter. There will be no domain controllers located elsewhere on the network.

Step 3: Design the sites

Contoso College's forest will be a single-site forest with all domain controllers located in the default site.

Step 4: Create site links

Since the forest will be a single site, there is no need to create any site links.

Fabrikam

Fabrikam is a global multibillion-dollar organization that has more than 250,000 employees and computers located at over 100 sites around the world. The business has its global headquarters in Chicago. There are four other major sites that link to the HQ and to which the smaller branch offices link: Asia-Pacific, North America, South America, and Europe. The small sites link to the hubs via links ranging from 64 kbps to T1 speeds; the hubs themselves connect to the HQ via a variety of higher-speed links. Some of the hubs are also interconnected.

Step 1: Gather background data for your network

Figure 14-7 shows the Fabrikam network topology at a high level. Links between datacenters are reliable although sometimes saturated. Multiple paths exist between the Europe, Asia-Pacific, and Chicago regional datacenters. WAN links to branch offices in

South America and some offices in Asia are often slow, saturated, and unreliable. These sites often also contain business-critical functions such as manufacturing operations.

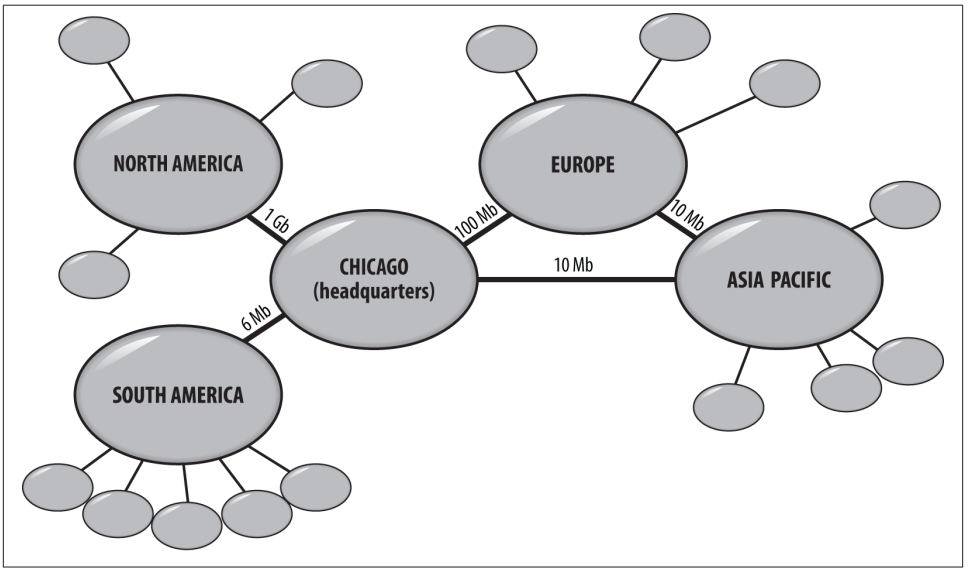


Figure 14-7. Fabrikam network topology

Fabrikam frequently opens and closes offices, and with over 100 offices, the subnet layout for the network changes frequently. To handle this, the Active Directory team receives a comma-separated values (CSV) file of subnets and their associated locations from the network team every night. An automated script processes the CSV file and ensures that Active Directory subnet information matches data from the network team.

Step 2: Plan the domain controller locations

Fabrikam plans to place writable domain controllers in each regional datacenter as well as the headquarters datacenter. The PDC emulator FSMO role holder will be placed in Chicago since Chicago is the most centrally located datacenter on the network.

Branch offices will receive a read-only domain controller if they meet any of the following criteria:

- At least 100 computers are located at the site.
- LDAP-integrated applications are hosted locally at the site and the WAN link is not reliable or fast enough to sustain the applications requirements.
- Business-critical activities such as manufacturing operations take place at the site.

Step 3: Design the sites

Fabrikam has numerous Active Directory–aware applications deployed across the enterprise, as well as automated processes for managing subnet objects. Every physical location on the network will have a corresponding site object deployed even if there is no domain controller hosted locally.

Step 4: Create site links

The site link topology will mirror the WAN topology shown in Figure 14-7. Site links between datacenters will have change notification enabled. All other WAN links will replicate at varying frequencies, depending upon reliability and requirements.

The cost attribute will be directly correlated to site link speed to account for multiple paths between some locations. Method C will be used to calculate the site link costs. Specifically, a reference bandwidth of 38400000000 bits per second (bps) will be used in the formula shown in Figure 14-8.

$$\sqrt{\frac{\text{Reference Bandwidth}}{\text{Link Bandwidth}}}$$

Figure 14-8. Inverse square root site link cost formula

Additional Resources

Microsoft has produced an outstanding guide for customers setting up a branch office deployment. A branch office deployment is a deployment with a large number of WAN sites that need to host local copies of Active Directory. An example is a large retail chain that has a central office and hundreds of retail outlets that are all tied together into the same forest. If you have to deploy this kind of infrastructure, you absolutely need to review the Windows Server 2003 Active Directory Branch Office Guide; you can find it on the Microsoft website. Even though the guide was written for Windows Server 2003, it is still worth reading.

Summary

Based on this chapter, you should have a good understanding of how to go about designing the site topology for your Active Directory deployment and also how to determine if a domain controller is necessary in any given site. Site topologies are leveraged by Active Directory for replication and also by various other applications, such as DFS and Exchange Server. Clients also rely on the site topology to locate a domain controller for authentication.

The Knowledge Consistency Checker (KCC) is the process that runs automatically on each domain controller to generate the underlying replication connections. Intersite replication connections are generated on one domain controller in a site that is designated the Intersite Topology Generator (ISTG) for that site.

Planning your site topology requires a good understanding of the underlying network as you consider your requirements for sites, site links, and site link bridges. Sites represent disparate network locations where you want to segregate replication and authentication traffic. Site links represent the connectivity between sites, and site link bridges provide the KCC with hints about transitivity between site links for generating the necessary connections for replication to occur.

The next chapter deals with how to update your designs to reflect the requirements for group policy objects in your organization.

CHAPTER 15

Planning for Group Policy

This chapter takes an in-depth look at group policy objects (GPOs), focusing on how to structure your Active Directory deployment effectively using organizational units and groups so that you can make the best use of the GPOs required in your organization.

Using GPOs to Help Design the Organizational Unit Structure

In Chapter 13, we described the design of the Active Directory organizational unit (OU) hierarchy. We also explained that other items have a bearing on that design. There are two key design issues that affect the structure of your OUs: permissions delegation and GPO placement. If you decide that your Active Directory infrastructure 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 domain-wide (rather than many GPOs on many OUs), your OU structure can be arranged 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 OUs in a manner that facilitates that administration. This doesn't have to be the case, but it makes it much easier to use OUs.

For example, if we have 1,000 users and 10 delegated administrators who each manage 100 users, we could put the 1,000 users in one OU and give each of the 10 administrators permission to modify only their 100 users. But obviously, this would not be an efficient system. Instead, it would be better to create 10 OUs and put 100 users in each, giving each administrator permissions over his particular OU. This makes much more sense, as the delegated administrator can be changed very easily, it is easier to report on access, and so on. Common sense and reducing management overhead are the overriding factors here. Either solution is feasible; one is just far easier to implement and maintain.

Permissions delegation is covered in more detail in Chapter 16.

The same fundamental facts apply to GPOs. If you are going to need to apply multiple policies to multiple sets of users, it will be easier to manage if you set up multiple OUs. However, this isn't always possible—for example, if the OU 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 Group Policy.

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 such as "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 subset of the user and computer accounts. If you aren't sure for some items, put the items in more than one category. You'll end up with items such as "Deploy financial applications to accountants in France" and "Restrict desktop settings in Europe."

Once you have the general policy areas constructed, you need to construct an OU structure that facilitates implementation of this policy design. At this point, you start placing computers and users in various OUs, and 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 permission restrictions, and what the OU 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 GPOs in a few ways based on the examples introduced in Chapter 13.

Guidelines for Designing GPOs

In this section, we provide guidelines that will help you meet two critical design goals:

- All policies should be applied quickly, so that users do not feel a significant impact due to policy processing.
- All policies should be as easy as possible to administer and maintain.

With these two goals in mind, let's take a look at the guidelines:

Design in a way that you feel comfortable with.
: As you'll see later in this chapter, it can be easier to do large designs by considering the user OU and computer OU structures separately. If you want to do them together and have a small enough network that you can do so easily, that's fine. If not, try treating them as separate Group Policy areas.

Restrict as best you can the number of policies that apply.
: In a perfect world, this wouldn't be important. But in the real world, the more policies you have, the more processing the client has to do in addition to its normal logon/boot up, and the longer it takes to complete the process.

  If you have multiple policies applying to an object from the same location in a tree, consider the possibility of collapsing them into a single object. If the number of policies you are applying during a logon/boot up is larger than you can get the client to process in a reasonable amount of time, you need to consider reducing or collapsing the policies. If you need to apply a large set of policies with many settings that extends logon to five minutes, but you feel that is acceptable to achieve this level of policy, that's fine.

  When it comes down to it, only you know what you can accept, and you will need to do your own testing in this area to satisfy your constraints. If you have to have a client logged on in less than four seconds, you have to work within that constraint. Microsoft likes to recommend nesting no more than 10 organizational units deep to make sure that you don't use too many GPOs. As we know, this isn't very helpful. Having one GPO applying at a site, one at the domain, and one at each of five organizational units means only seven GPOs. Applying 10 at each level is 70. So, it's not only how deep you nest your organizational unit structure that matters, but also how many policies you apply at each level.

Use security groups to tailor access.
: While you can set up ACLs to allow or deny application of policy to an individual user or computer, it makes more sense to use groups to do this whenever you can. Using groups lets you keep all policy access in one object and can make complex systems much more manageable. To keep troubleshooting straightforward, try to avoid the use of the deny capability and stick with explicitly allowing groups.

Limit the use of block inheritance/enforce policy.
: You should be very cautious of blocking inheritance at locations in the tree unless you are quite sure that this is the right way to solve your problem. The repercussions from a simple blocking of inheritance can spiral quickly as you encounter areas of a policy that need to override the block. Your well-designed system can become quite difficult to maintain if you block and override regularly. This is not to say that you should never use these options; just exercise caution in their use.

Collapse the organizational unit design.
: If you wish, you can collapse your OU design and make more use of groups to govern access to specific policies. These are both perfectly valid solutions, and you should use whichever one you are more comfortable with. Remember the axiom that the more you collapse the OU structure while maintaining or increasing the number of GPOs, the greater your need will be for control via groups.

Avoid using cross-domain GPO links.
: If you link GPOs across domains, the client must contact a domain controller in the GPO's domain to download information about the GPO. Unless you have very fast links between the client and the foreign domain with enough available bandwidth, you should duplicate the functionality of the GPO in the target domain instead of depending on cross-domain linking.

Prioritize GPOs.
: Remember that it is possible to prioritize the application of multiple GPOs at the site, domain, or OU level. This ordering of the application of policies allows you to add useful options to the administrator's toolkit. For example, if for a group of users you need to reverse specific settings that are being applied by default as part of a larger set, you can create a new GPO with ACLs for this group that apply in the priority list to unset all the previous settings. This solution allows you to override a selection of previous settings without creating two GPOs, one with settings for everyone and one for just this group. The former solution allows you to add settings to the main GPO later and still have them apply to everyone, without needing to also add them to a second GPO. Prioritizing GPOs can be very useful.

Be cautious with loopback mode.
: Loopback mode is a very useful tool, but this is another technology that you need to approach with caution. As it can result in a completely different set of policies (replace mode) or a very large number of policies (merge mode) being applied to your users, you need to take great care to ensure that the policy received by a user is the one you expect.

Thoroughly test WMI filters.
: If you are using Windows Management Instrumentation (WMI) filters, be sure to test the queries thoroughly before releasing them to production. If you use an inefficient query or one that is very resource-intensive, it could cause significant

delays during GPO processing. Consider using the free WMI filter validation utility available from this website to measure the impact of WMI filters.

Use a test environment to validate GPO changes.
: We always recommend creating a test environment with representative test clients in order to test the impact of GPOs before deploying them in production. No GPO should ever be applied to production users or computers unless it has been fully tested.

Choose monolithic or segmented GPOs.
: While we would recommend keeping similar settings—or all settings relating to a particular item—in the same GPO, there is nothing stopping you from having only a few huge GPOs as opposed to a number of smaller GPOs. If you go for the monolithic approach, you'll process fewer GPOs, which is obviously faster; however, delegation is not as easy due to the fact that the policies contain so many settings. Segmented GPOs allow easier delegation, but having more policies can impact performance. Mix and match the two to a level that you are comfortable with and that works for your network.

Design Examples

Let's take a look at the three sample organizations that were introduced in Chapter 13. In this section, we'll explore one possible Group Policy design for each organization, given their individual requirements.

Tailspin Toys

Tailspin Toys is an organization that employs approximately 3,000 people across North America. There are approximately 4,000 client computers and several hundred servers spread across the Tailspin Toys locations. Office workers are located in a number of cities across the United States and Canada, as well as in home offices. Manufacturing operations take place in Mexico. The IT Department located in the San Francisco headquarters office manages IT assets across the entire organization.

Tailspin Toys manages almost all of the objects in Active Directory centrally. Since there is limited need for complex permissions delegations, Tailspin Toys focused on Group Policy when designing its OU structure. After reviewing the organization's needs, a location- and object-based OU structure was developed. This allows policies to be targeted to physical locations as well as specific types of objects (e.g., desktop computers versus laptops).

Tailspin's need to manage the desktop experience on end user office workstations is relatively limited. There are a handful of global settings that are enforced across the organization, and from time to time a need to map a drive or install a printer at a specific location arises. Figure 15-1 shows the GPO design for the organization's end user computing environment. The basic premise of the design relies heavily on inheritance, linking GPOs at the highest possible point in the OU structure. User and machine settings are separated into separate GPOs for ease of understanding. In addition to the global policies, a country-specific policy can be applied as necessary, as well as optional location-specific policies at the user, computer, or computer type (e.g., desktops, laptops, or kiosks) level. The name of each policy is prefixed with an identifier for the location or locations affected by that policy.

In the manufacturing organization, the desktop experience must be tightly controlled in order to ensure that users cannot negatively impact the manufacturing process. Additionally, requirements vary from site to site and even within a site based on the roles of the computers on the manufacturing shop floors and the presence of multiple production lines. A strict user experience must be maintained regardless of the user who logs onto a desktop. The policy structure must also support multiple policy profiles for different sets of machines within the same OU.

Figure 15-2 shows the Tailspin Toys approach to this problem. In this design, a global computer policy is linked to the Manufacturing OU. This policy enables loopback processing in replace mode. Leveraging loopback processing ensures that user-specific settings can be applied to manufacturing machines regardless of the location of the user in the OU structure and without concern for policies applied to that specific user. In sites with only one policy requirement, a single policy named for that site code is linked to the site's OU. In sites such as the SFO01 site shown in Figure 15-2, each separate policy is restricted with security filtering to control the machines that the policy applies to. In all cases, policy names are prefixed with *Manufacturing* and then the site code for the location the policy affects so that manufacturing-specific policies can easily be distinguished from policies that apply to end user workstations.

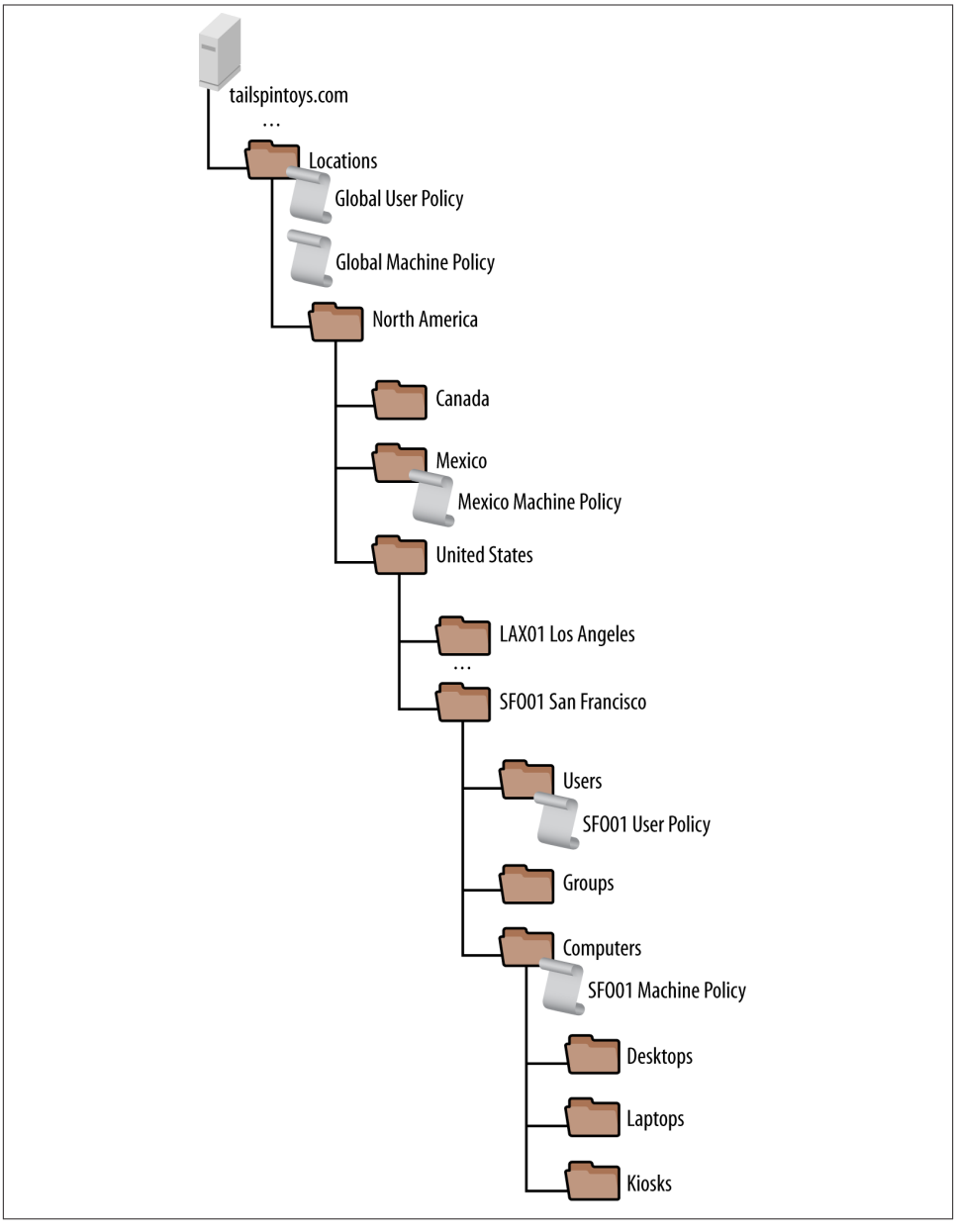


Figure 15-1. Tailspin Toys end user computing GPO design

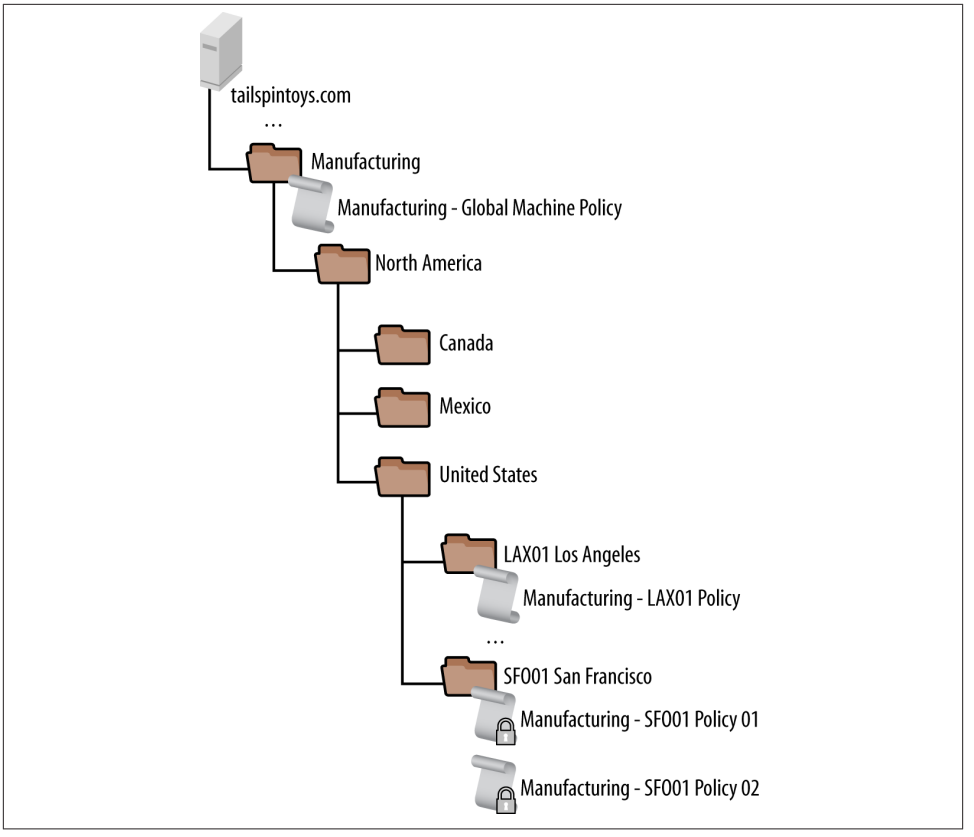


Figure 15-2. Tailspin Toys manufacturing GPO design

Contoso College

Contoso College is a higher education organization with approximately 3,500 employees, 30,000 students, and 5,000 computers. All users and computers are located at a single campus. Administration of IT resources is decentralized, with individual departments maintaining large-scale autonomy over the computers and resources in their areas. In an effort to provide a centralized, sustainable, and distributed directory service, central IT is implementing an Active Directory forest that supports automatically managed user accounts and groups as well as the ability for delegated administrators to receive OUs whose resources they can manage as they wish.

Contoso needed to satisfy a number of sometimes competing demands when planning its OU structure. The primary need was for a domain that was manageable by central IT resources while also being flexible enough to meet the needs of delegated administrators in various departments at the college. Given the decentralized structure, Contoso College decided to take a relatively hands-off approach to Group Policy. In its

approach, only settings that were required to be globally enforced in order to meet organization-wide requirements and security audit and compliance requirements were enacted. To ensure that delegated administrators do not override policy settings that are mandatory across the organization, central IT sets the Enforced flag on the global policies.

Individual department administrators are placed in the Group Policy Creator Owners group and given the ability to create and manage GPOs as they see fit. Security delegations in the OU structure limit delegated administrators to only linking policies in their respective OU structures. Since all users are located in a central People OU, individual department administrators use loopback processing to apply user-specific settings depending on where the user logs on.

Fabrikam

Fabrikam is a global multibillion-dollar organization that has more than 250,000 employees and computers located at over 100 sites around the world. The business has its global headquarters in Chicago. There are four other major sites that link to the HQ and to which the smaller branch offices link: Asia-Pacific, North America, South America, and Europe. The small sites link to the hubs via links ranging from 64 kbps to T1 speeds; the hubs themselves connect to the HQ via a variety of higher-speed links. Some of the hubs are also interconnected.

Fabrikam's management structure is regionalized, with each geographical unit running itself as an independent business as part of the global organization. The top level of the management structure is located at headquarters, which sits above the four hubs. Even though Chicago could be considered within the North America region, the organization is not structured that way. In fact, IT personnel in Chicago oversee the hubs in terms of selecting the administrators and determining how the network is to be structured. Corporate IT policy dictates that branches that have more than 500 people have their own local administrator, backup support, and helpdesk staff. Branches with fewer than 500 people are managed by the administrators of the hub to which they connect.

Group policy requirements at Fabrikam are driven by two needs:

- Security audit and compliance mandates
- Site-level user experience configuration requests

Outside of security audit and compliance mandates, Fabrikam has no need or desire to manage the user experience beyond out-of-the box settings. To this end, the organization elected to create GPOs at the global and country levels to enforce compliance settings. Delegated administrators at individual sites can request a GPO for their site along with a brief description of the purpose of the policy, a business justification, and a security group to delegate policy editing rights to. Assuming the policy is approved,

Active Directory administrators will create and link the policy and then delegate management rights to the requested group. Figure 15-3 shows Fabrikam's GPO design.

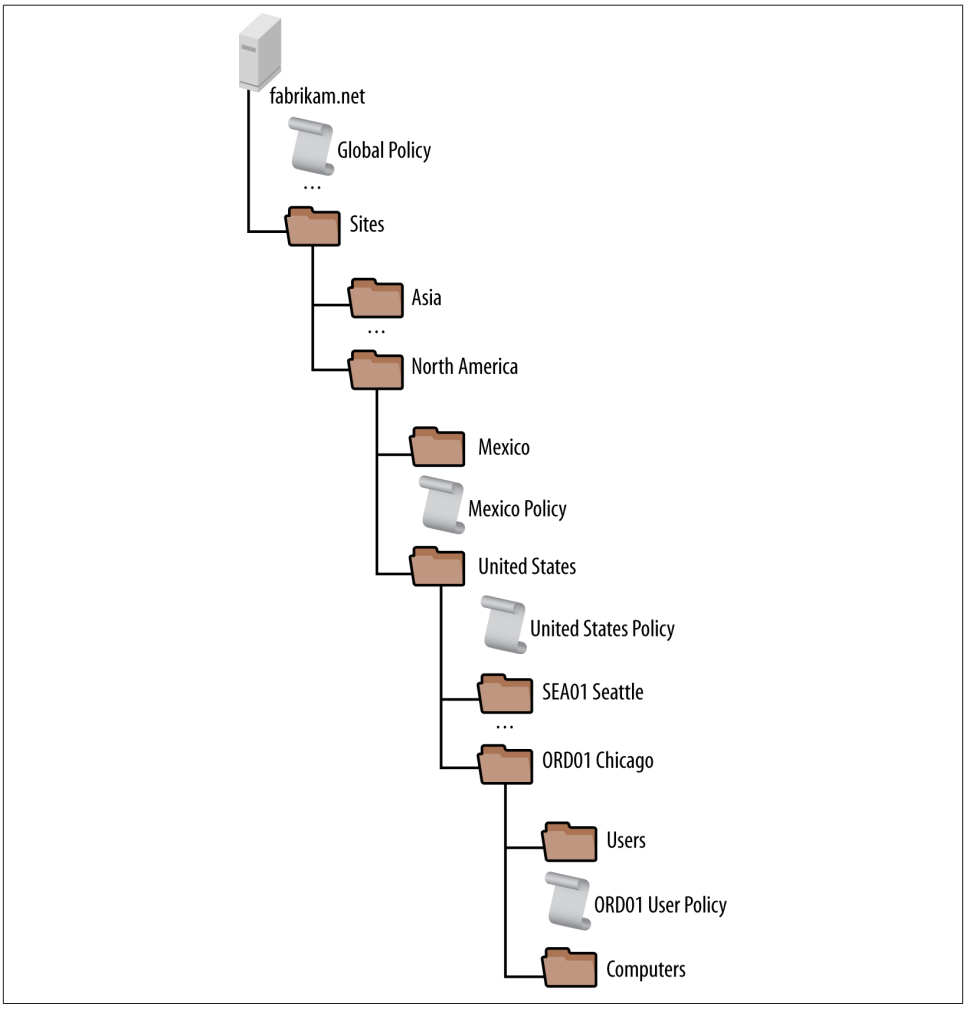


Figure 15-3. Fabrikam Group Policy design

Summary

One of the big selling points of Active Directory has always been Group Policy. In this chapter, we expanded on the information presented in Chapter 11 to cover strategies for designing your Group Policy deployment.

CHAPTER 16

Active Directory Security: Permissions and Auditing

Permissions can be set in Active Directory in much the same way they are set for files. Although you may not care that every user in the directory 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.

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 system administrator. The GUI that Microsoft provides is effective for simple tasks, but more cumbersome for managing complex permissions. Management of Active Directory permissions is also supported by Active Directory Service Interfaces (ADSI), which opens up a whole raft of opportunities for you to use scripts to track problems and manipulate access simply and effectively. Finally, Windows PowerShell and the *DSACLS* utility allow 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 the 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.

Dynamic Access Control (DAC) is a major new feature in Windows Server that Active Directory helps bring to life. Configuring your forest to support claims-based authorization to resources as well as new features such as compound identity and central access

policies is a task that will require close coordination of Active Directory administrators along with other members of the IT organization and the business as a whole.

Permission Basics

Figure 16-1 shows the basics. Each object stores a value called a *security descriptor* (SD), in the `nTSecurityDescriptor` attribute. This attribute holds all the information describing the security for that object. Included with this information are a flag indicating whether or not the security descriptor is protected against inheritance and two important collections called access control lists (ACLs), that hold the relevant permissions.

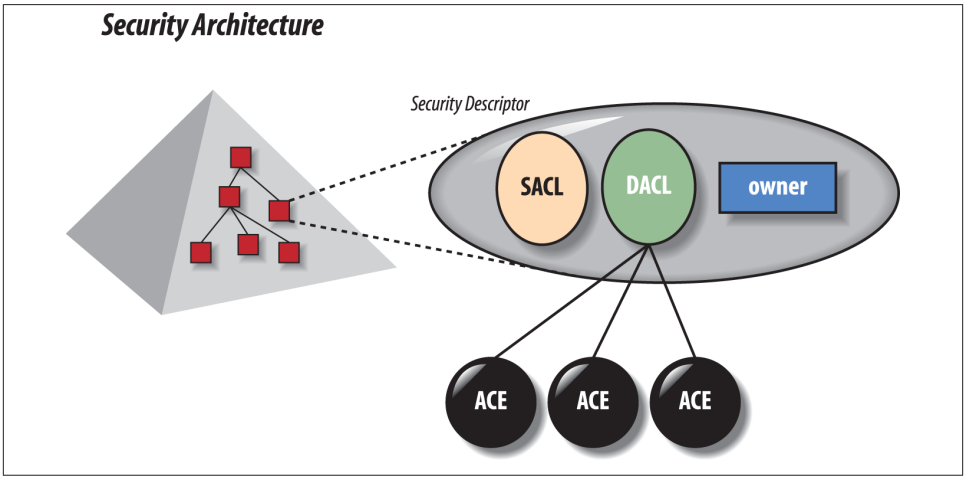


Figure 16-1. Active Directory security architecture

The first ACL, called the *System 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* (ACEs) that correspond to individual audit or permission entries.

Audit and permission ACEs can apply to the object as a whole or to the individual properties of the object. This allows an administrator to control all aspects of access and auditing. Audit ACEs are discussed further later in this chapter, but we will briefly document permission ACEs in the following section so that other permission-related topics discussed first are more readily understood.

Permission ACEs

Each permission ACE is made up of several pieces of information:

Trustee
: The SID of the user or group to which the ACE applies, such as the SID for the group *COHOVINES\Domain Admins*.

ACE Type
: Determines whether the ACE is an *allow* or a *deny*.

Object Type
: The `schemaIDGUID` for the attribute or object class that the ACE applies to or the `rightsGuid` for the property set, validated write, or extended right that the ACE applies to (such as the `member` attribute, `Personal Information` property set, `Change Password` extended right, or `user` objects). For Delete or Create Child Objects permissions, the `objectType` should be configured to the `schemaIDGUID` of the object class delegated.

Inherited Object Type
: The `schemaIDGUID` for the types of object that the ACE applies to when an attribute, property set, or validated right is specified or when the ACE is inherited—e.g., `user` objects.

Access Mask
: A bit flag that describes the type of access, such as Read, Write, List, Create, Delete, Control Access, etc. See Table 16-1 for more detail.

Flags
: There are actually two different fields for flags. The flags specify inheritance settings such as ACE is inherited, ACE is allowed to be inherited, ACE is not inheritable, etc.

Table 16-1. Contents of an ACE's properties

Name of the property	Sample value to be stored
`Trustee`	Names the security principal that is to have the permission.
`AccessMask`	Gives write access to a specific property.
`AceType`	This is an allow permission.
`AceFlags`	The permission is inherited only and does not apply to this object. Child objects inherit this ACE.
`Flags`	Both `ObjectType` and `InheritedObjectType` are set.
`ObjectType`	This is the `schemaIDGUID` of the `description` attribute.
`InheritedObjectType`	This is the `schemaIDGUID` of the `User` class.

Every permission granted in Active Directory is based on these ACE structures. Objects and properties are only revealed to users who have the appropriate permissions on those objects. Any attribute that doesn't have an ACE specifically granting the requested access to a trustee is implicitly denied. The security descriptor also allows an administrator to specifically deny access with a deny ACE, but this isn't needed unless you are attempting to override another more generic ACE that is granting access.

This permission mechanism allows different users or groups of users to have completely different and very granular access to an object. For example, all users might be granted read access to the telephone number and email properties, but only a subset will have access to modify the description.

Property Sets, Validated Writes, and Extended Rights

Microsoft has introduced several new "tools" that are not present in the NTFS ACLs. These tools are called property sets, validated writes, and extended rights. Instead of inserting the `schemaIDGUID` into the Object Type field of an ACE, you can insert the `rightsGuid` attribute of the property set, validated write, or extended right object. These objects are all stored in the `cn=extended-rights` subcontainer of the Configuration container.

Property sets, validated writes, and extended rights are all stored as `controlAccessRight` objects. You can determine what specific type of `controlAccessRight` they are by looking at the `validAccesses` attribute of the object with any utility that allows viewing of all attribute values on an object, such as LDP or ADSI Edit.

Property sets have a value of `48`, validated writes have a value of `8`, and extended rights have a value of `256`. For more information, reference this website.

Property sets are collections of attributes that can be referenced in a single permission ACE. The big win here is that you could assign one ACE to an SD and have it grant access to 5, 10, 20, or more attributes. The savings in SD size should be immediately obvious: a single ACE that can substitute for 20 different ACEs is a good data reduction ratio. There are several predefined property sets in a default forest; some of these are Personal Information, Public Information, Web Information, and Account Restrictions. You can also add your own property sets as desired. Exchange adds a number of property sets to ease delegation, including Exchange-Personal-Information and Exchange-Information.

While property sets are a great boon for securing the directory in a manner that is simpler than could otherwise have been achieved, the implementation does suffer a major shortcoming. *An attribute can only be part of a single property set*. Unfortunately, many of the base schema attributes are already included in existing property sets; while most of these can be successfully removed from those property sets or moved to other property sets, you can never be sure doing so won't break some application. It is generally much safer to stick to managing property sets comprised of your own custom attributes that you add to the directory.

Validated writes are writes that go through additional verification. You cannot modify the validated writes, nor create your own; what comes with AD is what you get. The only validated writes that are currently defined are:

- Validated write to service principal name
- Validated write to DNS host name
- Add/Remove self as member

Extended rights are the mechanism used to delegate special operations such as password changes or password resets. While you can add additional extended rights to the directory, you cannot create extended rights that are enforced by the directory. Any new extended rights that you create will simply be additional permissions that can control an application you write that knows how to use the permissions. Your primary use of these extended rights will generally be to delegate to admins the right to set passwords on user objects with the Reset Password extended right.

Inherited Versus Explicit Permissions

Many Windows administrators are used to quoting the mantra, "Deny overrides everything." Fortunately (or maybe unfortunately, depending on what you are trying to accomplish), Active Directory ACLs are a little more complicated than that. You need to specifically be aware of inherited versus explicit permissions. *Explicit permissions* are permissions that are directly applied to an object. *Inherited permissions* are permissions that are applied at some level of the tree above the object and "flow down" to the object and its children. When working with inherited and explicit permissions on an object, *a deny doesn't necessarily override a grant*.

The rules for what access will result from a set of inherited and explicit ACEs are easiest to understand when taken in steps:

1. Deny ACEs override grant ACEs of the same type (inherited versus explicit) and application point in the directory tree.
2. Explicit ACEs override inherited ACEs.

3. Inherited ACEs are hierarchical (e.g., an inherited deny applied to a container will be overridden by an inherited grant applied to containers nested below it in the same tree).

Most simply, the closest ACEs to an object will dictate the access for that object. If you use an inherited deny to prevent access to some attribute on objects in an OU and the deny isn't effective, look for inherited grant permissions applied further down the branch or for explicit grant permissions applied on the objects themselves.

Default Security Descriptors

Every object defined in the schema has the attribute `defaultSecurityDescriptor`. This attribute is the *Security Descriptor Definition Language* (SDDL) format of the object's default SD. It will be applied to any object created without an SD specified during the object creation. These default permissions are rather extensive in Active Directory and are composed entirely of explicit ACEs, so that they override any inherited denies (this tends to make denying access to specific attributes a little more challenging than it probably should be). The DACL defined by the default security descriptor for the user object is listed in Table 16-2.

Table 16-2. DACL for the default security descriptor on the user objectClass

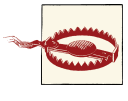
Trustee	Permission
Domain\Domain Admins	Full Control
Builtin\Account Operators	Full Control
System	Full Control
Self	Read Permissions List Contents Read Property List Object Read/Write Personal Information Read/Write Phone and Mail Options Read/Write Web Information Change Password Send As Receive As
Authenticated Users	Read Permissions Read General Information Read Public Information Read Personal Information Read Web Information
Domain\RAS and IAS Servers	Read Account Restrictions Read Logon Information Read Group Membership Read Remote Access Information

Trustee	Permission
Domain\Cert Publishers	Read/Write userCertificate
Builtin\Windows Authorization Access Group	Read tokenGroupsGlobalAndUniversal
Builtin\Terminal Server License Servers	Read/Write terminalServer
Everyone	Change Password

Permission Lockdown

You can see from Table 16-2 that in a default Active Directory installation members of the Authenticated Users group have access to quite a bit of information. There is even more information available if you have added Authenticated Users to the Pre-Windows 2000 Compatible Access group, as this allows them to see all properties of groups, users, and `inetOrgPerson` objects. Open access to this information is a sore spot for many companies who don't want employees to have quite that much information available to them; maybe they want to restrict access to employee phone numbers, or addresses, or other personal information.

Removing the ability to see this information can be difficult to accomplish, especially for the items granted through explicit property set ACEs. In addition, any modification you make needs to be thoroughly tested to verify it doesn't break any of your line of business applications. Microsoft Exchange is particularly sensitive to permission lockdowns, and if you take away the Authenticated Users group's access to some attributes, you may need to re-grant the permissions back to the Exchange servers. ADSI is another common tool that is very sensitive to changes to the default permissions. You may find ADSI calls fail with inexplicable errors if assumed permissions have been removed.



You cannot permanently lock down the directory from Domain Admins. At any point, anyone from that group can take ownership of any object (if they don't already have it) and rewrite the security descriptor to grant access to Domain Admins or any other security principal. There should be very few Domain Admins in your forest. We generally recommend a total number in the realm of five or fewer, *even for the largest organizations*.

Once you have decided to proceed with the lockdown, your next step is to decide how to accomplish it. The "how" and the resulting workload and impact depend entirely on the attribute in question and how the access is being granted.

By far, the simplest case is when the access is granted through inheritance. You can either remove the inherited grant if it is just the one attribute, or add an inherited deny for the attribute or attributes you want protected. An example of an inherited permission you may want to remove is read access to the `employeeID` property. Authenticated Users are granted read access to `employeeID` through an ACE applied to the root of the domain

that grants the Pre-Windows 2000 Compatible Access group permission to read all properties of users. Inserting an inheritable deny read ACE on the root of the domain or other container for Authenticated Users will effectively block the inherited grant read access to this attribute. Read access can be re-added further down the tree beneath the deny ACE if desired, with either an additional explicit or inherited grant read access ACE.

If the access is granted through an explicit ACE, locking down is considerably more involved and may actually be so difficult it isn't feasible. There are several mechanisms available for locking down explicitly granted access:

- If the access is granted through a property set, modify the membership of the property set involved.
- Change the default security of the object class definition in the schema to remove the explicit Authenticated Users permissions, or add a deny ACE for the specific attribute and then "fix" the ACLs on every instance of the object in existence in the directory. Any new objects will be created with the new security descriptor.
- Strip the explicit grant ACE from every instance of the object in the directory. You will need to keep doing this as new instances of the object are created, or if someone resets the ACL to schema defaults on any of the objects.
- Add an explicit deny ACE on every instance of the object in the directory. Like the previous solution, this requires "fixing" new instances of the objects that are created or reset to schema defaults.
- You may be able to use the confidentiality bit. This capability is described in the following section.

The Confidentiality Bit

After five years of customers wrestling with the directory lockdown difficulties previously outlined, Microsoft added a feature called "confidential attributes." As part of this feature, a new `searchFlags` bit called the *confidentiality bit* was added. This is bit `7`, which has the value `128`. When the bit is enabled, the attribute is considered to be confidential. Only users with both Read Property permission *and* Control Access permission for the attribute on an object can view the attribute. This means anyone with Full Control or All Extended Rights (Control Access for object) or Control Access for the specific attribute. Assuming default permissions, this means access would be granted to:

- Builtin\Administrators (inherited Full Control)
- Builtin\Account Operators (explicit Full Control)
- NT Authority\System (explicit Full Control)
- Domain\Domain Admins (explicit Full Control)
- Domain\Enterprise Admins (inherited Control Access)
- Object Creator\Owner (explicit Control Access)

As you can see, this would appear to give us a great new solution to the issues mentioned previously with explicit permissions. In theory, you can simply mark any troublesome attributes as confidential, and normal users will no longer be able to see them. This is a great plan, but there is still a challenge: you cannot set the confidentiality bit on Category 1 attributes, which are more generically known as *base-schema attributes*. So, many of the attributes companies may be trying to lock down can't be locked down with this capability.

Unfortunately, there is not an easy mechanism to grant the Control Access right with a GUI tool. The best available tool is LDP. The ACL editor in ADUC and ADSI Edit only supports granting Control Access over all the attributes of an object (via the All Extended Rights permission), not just a single attribute. For more information on managing ACLs with LDP, refer to Chapter 20.

Protecting Objects from Accidental Deletion

A common problem Active Directory administrators have faced is when an object or an entire OU tree is accidentally deleted. Usually this happens because of a careless mistake made with the Active Directory Users and Computers tool. The ensuing recovery process often involves authoritative restores and business downtime. Newer versions of ADUC include a checkbox in the MMC called "Protect object from accidental deletion." Behind the scenes, this checkbox simply adds an ACE to the ACL for the object it is applied to and prevents deletion. Consequently, this will prevent a tree from being deleted, since its parent cannot be deleted. In order to delete that object or tree, the administrator must first uncheck the checkbox or remove the ACE manually.

In order to see the "Protect object from accidental deletion" checkbox, you must first select View→Advanced Features in ADUC. Once you have done this, you can right-click an object in the directory and select Properties, and then access the Object tab. Figure 16-2 shows the People OU being protected in ADUC, and Figure 16-3 shows the People OU being protected with ADAC.

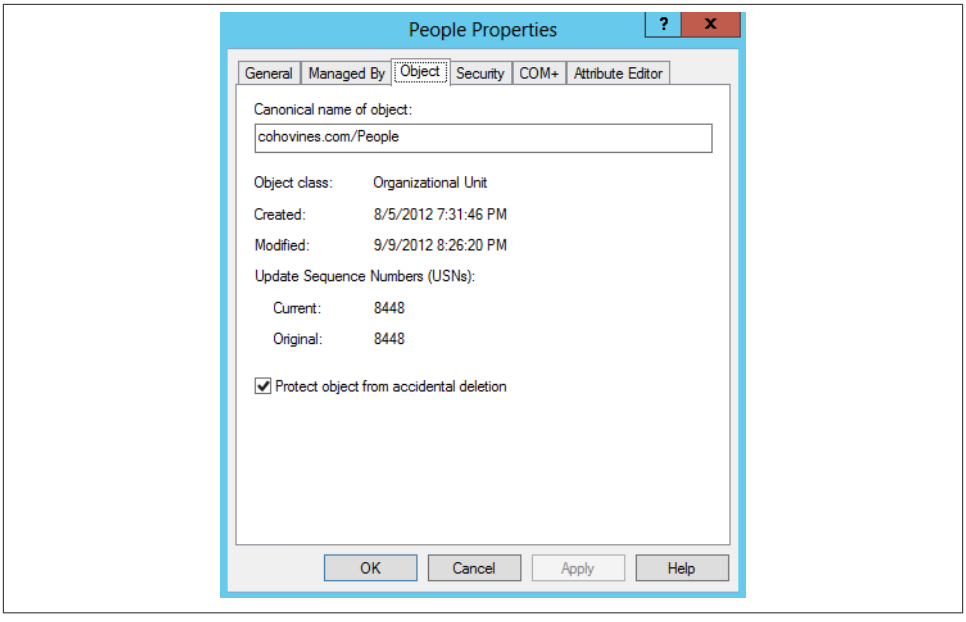


Figure 16-2. Protecting the People OU with ADUC

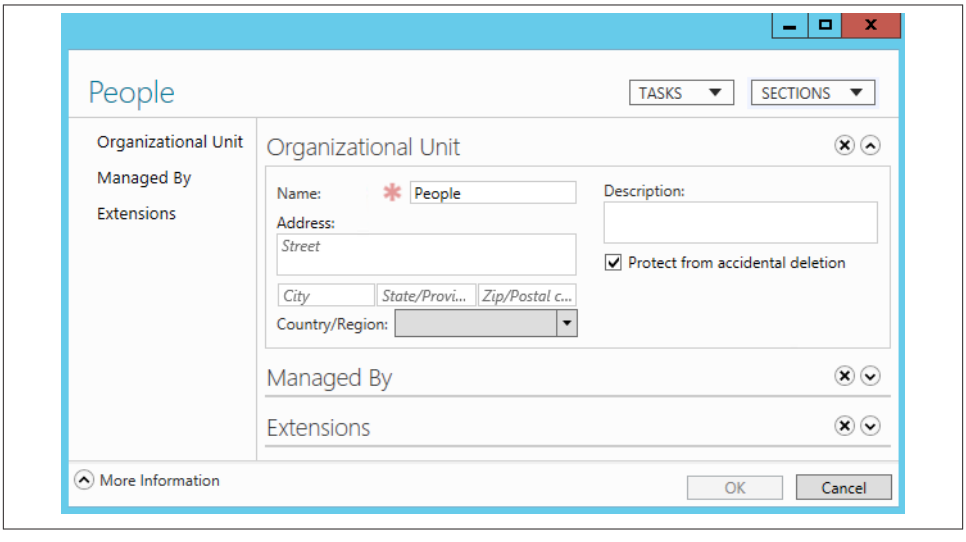


Figure 16-3. Protecting the People OU with ADAC

If you try to delete an object that is protected from accidental deletion, you will receive an error similar to the one in Figure 16-4. In order to delete the object, you must go

back into the properties and uncheck the “Protect object from accidental deletion” (“Protect from accidental deletion” in ADAC) checkbox.

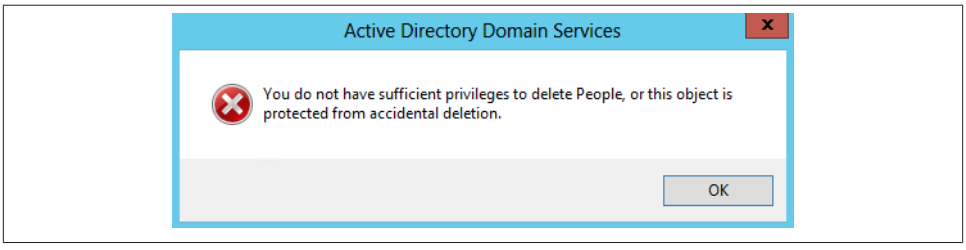


Figure 16-4. Error message when trying to delete a protected object

Figure 16-5 shows the ACE that was added to the ACL for the People OU in order to protect it. Notice that the well-known Everyone security principal has been denied the rights to delete the object or the subtree.

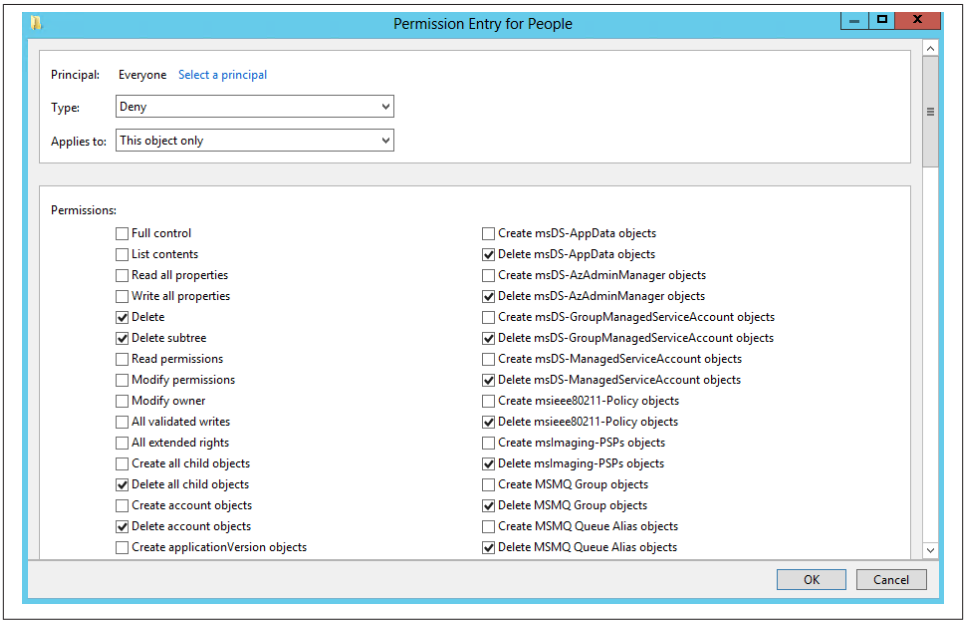


Figure 16-5. ACE created by protecting the People OU

Using the GUI to Examine Permissions

To access the permissions for any object, select the Active Directory Users and Computers MMC snap-in 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 select View→Advanced Features. 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 16-6 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 viewing. 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. For example, the general permission called Read translates to specific permissions like Read All Properties and List Contents, as we will show later.

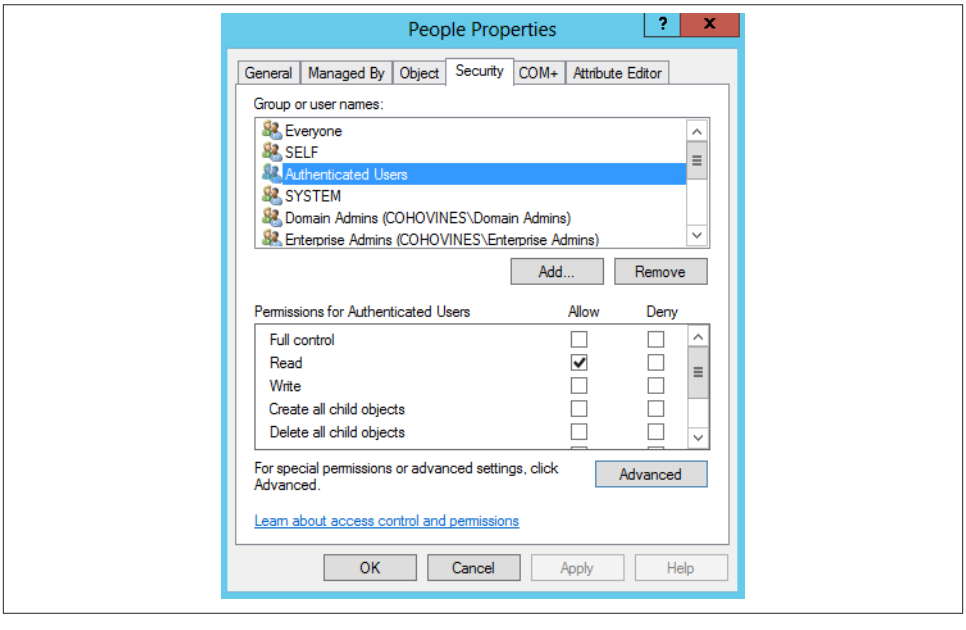


Figure 16-6. Security properties of an object

While the Advanced Security Settings window shown in Figure 16-7 gives only slightly more information than the previous window, it serves an important purpose: it is a gateway to some of the lowest, most granular permissions. The Advanced window allows you to view the globally set permissions from Figure 16-6 as well as a brief summary of the advanced permissions that may be set for each object. While the Principal and Access columns effectively duplicate information from Figure 16-6, the Type field shows whether the permissions to the object for this user or group are allow or deny permissions. If a group has some allow and some deny permissions, two entries are recorded in this window. The "Inherited from" column allows you to see what object, if any, the permission was inherited from. The "Applies to" column usefully indicates what the permission applies to; it could be to this object only, to the object and all subobjects, or just to an individual property (say, the `telephoneNumber` property of a user object).

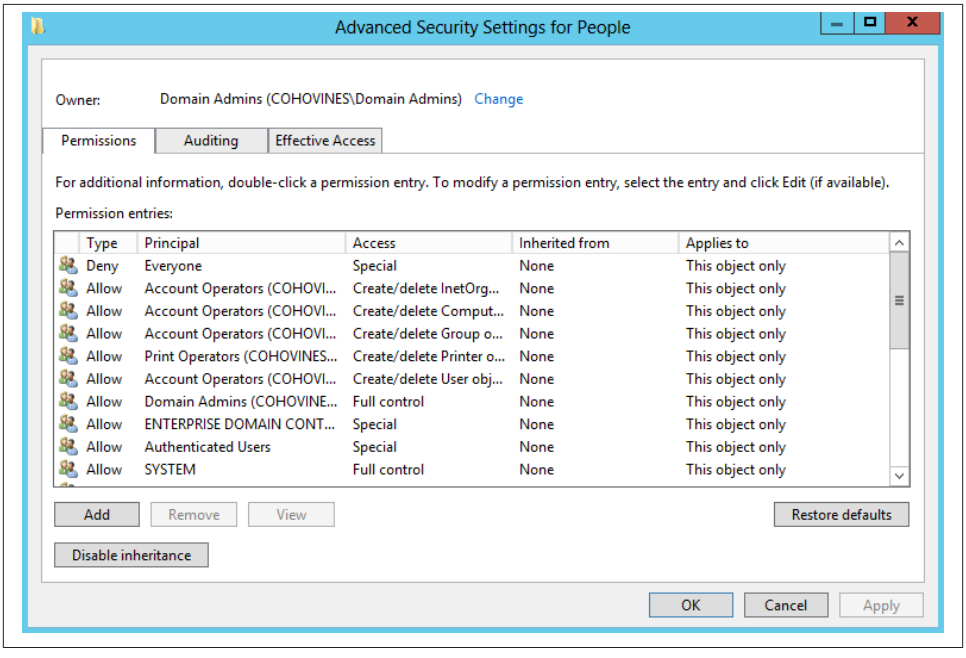


Figure 16-7. Advanced security Settings for an object

The permission dialog boxes contain quite a lot of information and are usually too small to see the information comfortably. This makes it painful to look at permissions, because you are constantly resizing the individual columns or scrolling the dialog back and forth. Unfortunately, there is no way to modify these dialog boxes to better display the information they contain.

You now have two choices to view the atomic permissions. You can click Add, which pops up a window allowing you to add a new user or group to those with permissions set on this object. Alternatively, you can highlight an existing user or group and click the Edit button. If you highlight a user or group or add one from the pop-up window, the next screen you see is the Permission Entry (PE) window, shown in Figure 16-8.

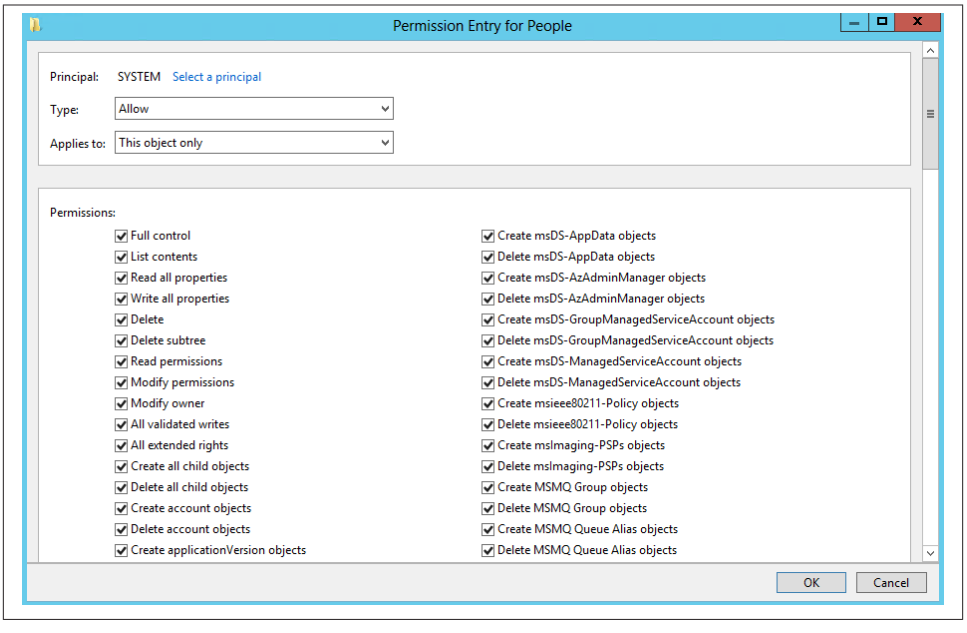


Figure 16-8. Permission Entry for an object

Until you know exactly what you are doing with permissions, we suggest that you use a test forest to experiment with permissions settings. The last thing you want to do is make a simple mistake with a built-in group or user and deny yourself access to the tree. If you create two test users and three test groups, put each user in a separate group, and then put both users in the third group, you will have the basis of a test system.

The object name is displayed in the title of the permissions editor window, with the name of the user or group that has permissions prominently displayed in the field at the top. The user or group then has permissions allowed and denied from the column entries. The entries in the window are relative and vary depending on the entry in the drop-down list under the heading "Applies to". What is not immediately obvious from this window is how large the drop-down box can actually get. Figure 16-9 shows this

nicely. If you look at the scroll bar, you will get an idea of how many items are currently not displayed.

To set a permission from the permissions editor window, pick where you want to apply the permission and then check the relevant Allow and Deny boxes, selecting OK when you're done. Since Microsoft has not provided an Apply button, you cannot specify a set of permissions to apply one area, click Apply, and then repeat the cycle until you are done with this user and group. You have to click OK, which means the window closes, whereupon you then have to click Add again, and so on. This is a tiresome problem if you are implementing multiple changes from a set of prepared designs, but one you have to live with if you choose to use the GUI to set permissions.

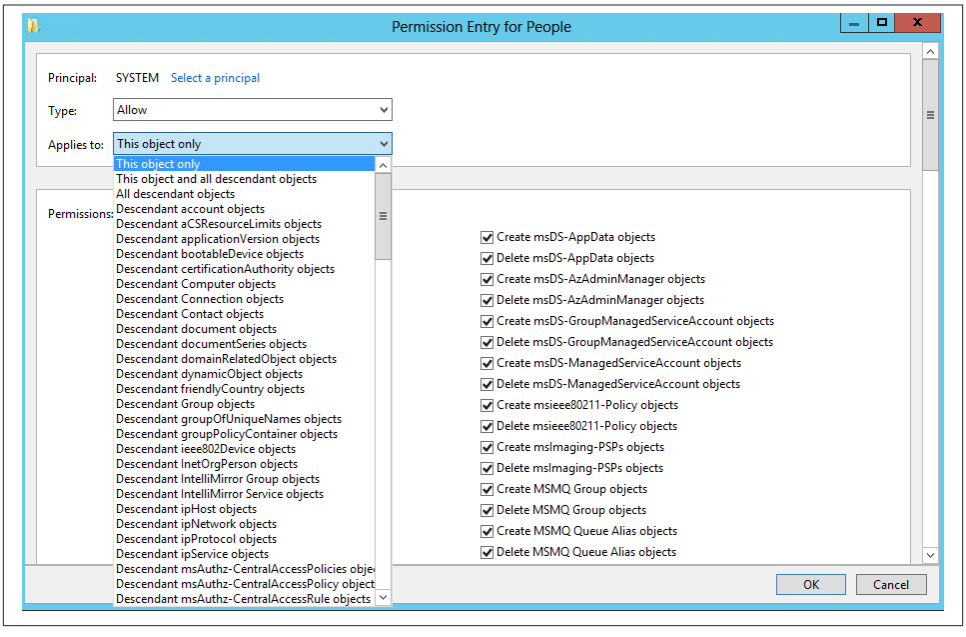


Figure 16-9. Permission Entry window showing the large number of targets to which permissions can be applied

Reverting to the Default Permissions

In Figure 16-7, you may have noticed the "Restore defaults" button at the bottom of the screen. This button allows you to revert the current permission set to the defaults, as defined in the schema for the `objectClass` of the object. If you click "Restore defaults" for an object on which you have not modified the permissions, you may notice that the list still changes. If you look more closely, you'll see that the inherited permissions were the ones removed. That is because inherited permissions are not defined as part of the default security of an object. Even if you then click OK to apply the permissions, as long

as the "Disable inheritance" button isn't pressed, the inherited permissions will still apply. Having the ability to apply the default permissions is a useful feature, especially for administrators who are trying to determine what changes have been made from the default installation.

Viewing the Effective Permissions for a User or Group

The Effective Access (or Effective Permissions in some versions of Windows) tab is available from an object's Advanced Security Settings screen (accessible in ADUC by viewing the object's properties, selecting the Security tab, and clicking the Advanced button). This screen allows you to select a user or group and determine its effective (or actual) access to the object, taking into account group membership and permission inheritance. Figure 16-10 shows the effective permissions for Authenticated Users on the People OU object. As you can see, Authenticated Users have List Contents, Read All Properties, and Read Permissions rights. All objects in the forest will inherit these permissions unless inheritance has been blocked. As you might guess, this is a significant feature that allows for much easier troubleshooting of permission problems. There are some limitations to be aware of, however.

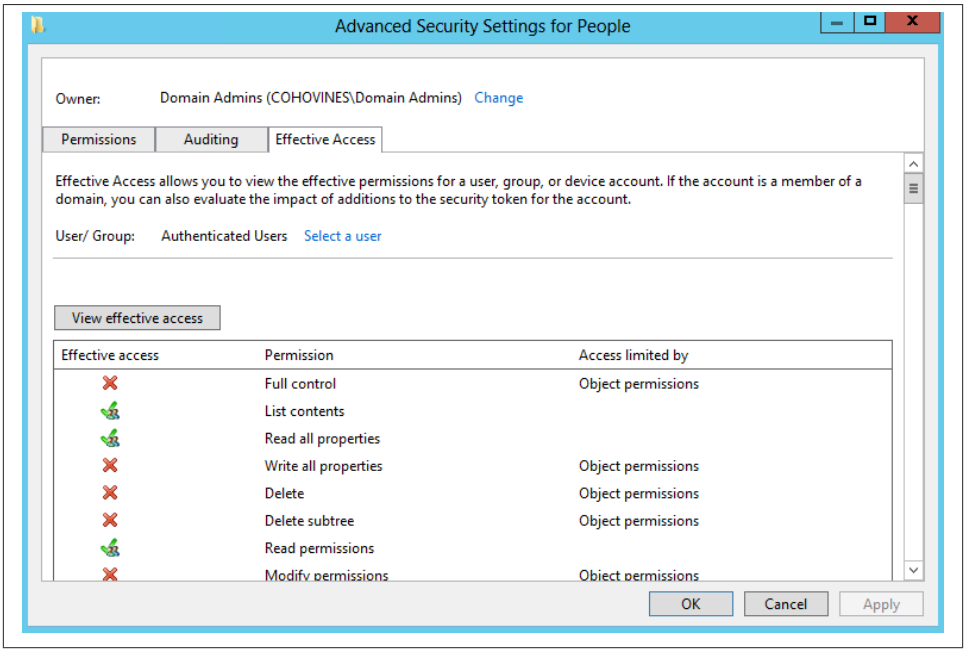


Figure 16-10. Viewing the effective permissions for Authenticated Users on the People OU object

The Effective Permissions tool provides only an approximation of the actual permissions a user or group has and does not take into account many of the well-known security principals, such as Anonymous Logon and Network. Another potential issue to be mindful of is that the user running the Effective Permissions tool must have the rights to read the group membership of the target user or group. By default, the Authenticated Users has group this right.

Using the Delegation of Control Wizard

To help with delegating permissions for objects in Active Directory, Active Directory Users and Computers comes with a *Wizard* called the *Delegation of Control wizard*. It is intended to allow administrators to delegate management of certain types of objects to key individuals or groups in the organization. It is activated by right-clicking almost any container in the tree and selecting the wizard from the pop-up menu. Builtin and LostAndFound are the two containers for which it does not work by default.

The wizard is useful only when you need to clearly apply general allow permissions to one or more object types below a container. It is not useful if you want to specify deny permissions (which it doesn't support), remove previously delegated control, delegate control over individual objects, or apply special permissions to branches of the tree. The wizard's great strength is its ability to set permissions and apply them to multiple users and groups at the same time. We use the wizard to set these sorts of permissions, although much less regularly than we do the standard GUI, since it is much more limited in what it can do. Scripting with ADSI also provides a solution here that is more adaptive to an administrator's own needs.

The wizard provides several screens for you to navigate through. The first is the welcome screen, which tells you what the wizard does. The second is an object picker for you to select which users or groups to delegate to. The third screen asks what task you wish to delegate control for in that container. Figure 16-11 shows this window.

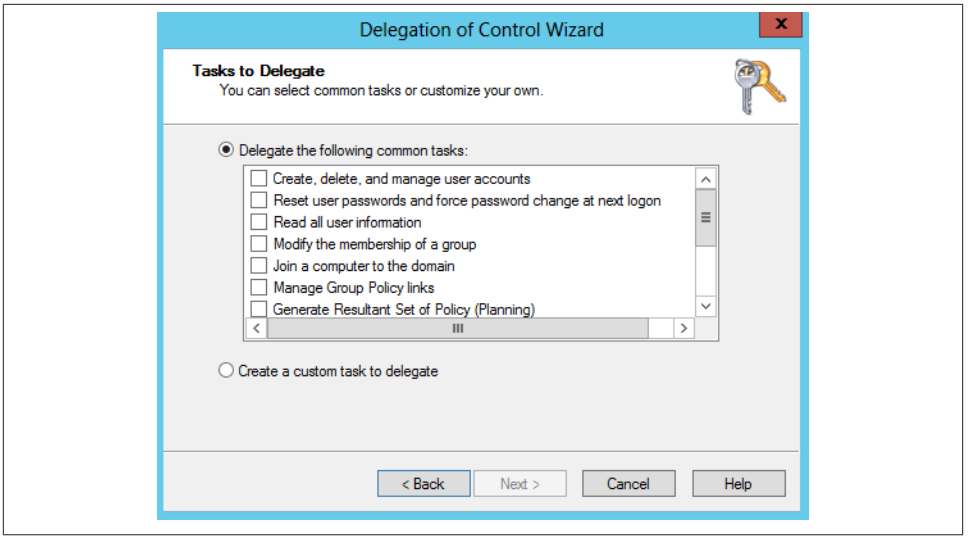


Figure 16-11. Delegation of Control wizard—task selection

The default is to delegate control for a specific task, and there are several to choose from. Since the list scrolled off the screen in Figure 16-11, we'll list them here:

- Create, delete, and manage user accounts
- Reset user passwords and force password change at next logon
- Read all user information
- Modify the membership of a group
- Join a computer to the domain
- Manage Group Policy links
- Generate Resultant Set of Policy (Planning)
- Generate Resultant Set of Policy (Logging)
- Create, delete, and manage inetOrgPerson accounts
- Reset inetOrgPerson passwords and force password change at next logon
- Read all inetOrgPerson information

If you choose the Custom radio button and click Next, an extra page opens, allowing you to specify individual objects. Figure 16-12 shows this.

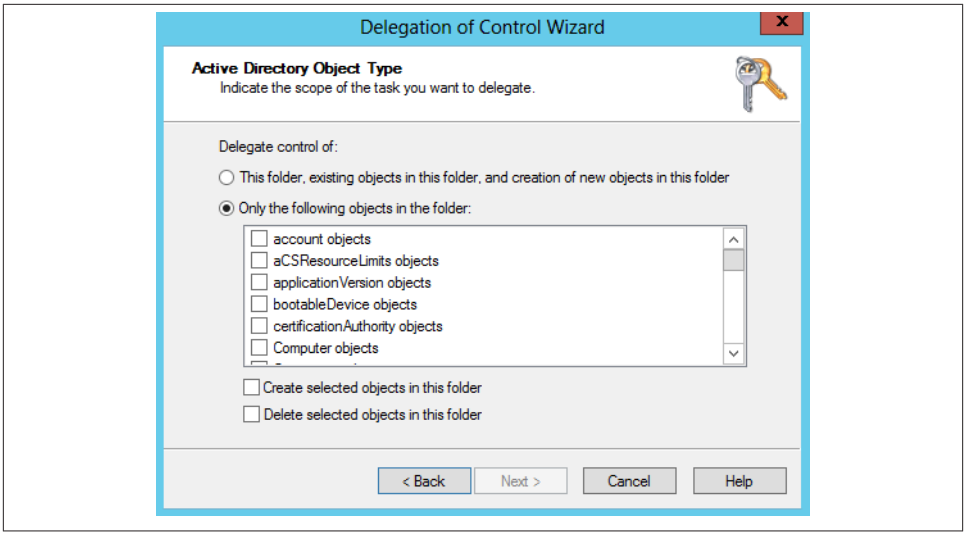


Figure 16-12. Delegation of Control wizard—choosing objects to delegate

If you want to delegate certain permissions to computer or user objects in a specific container or branch, you can do it from here. The next screen of the wizard allows you to specify what permissions you wish to assign for the selected users/groups. Figure 16-13 shows this screen.

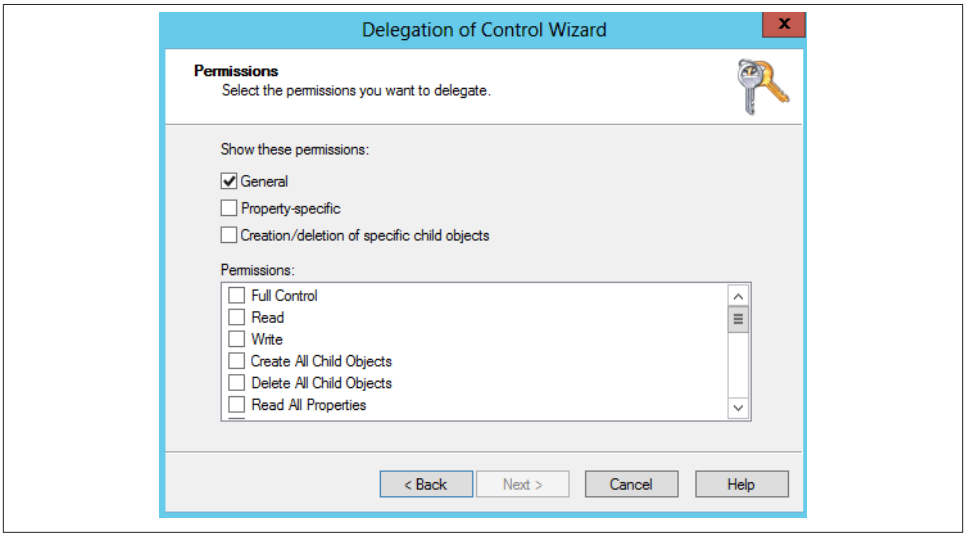


Figure 16-13. Delegation of Control wizard—access rights selection

When the window opens initially, only the first checkbox is checked. As you click each of the other boxes, the list of specific permissions that you can delegate becomes very large as it encompasses all of the permissions that you could potentially delegate. Finally, the last screen of the wizard summarizes the previous answers and allows the user to go back, cancel, or finish and grant the permissions.

However, just as the permissions listed in the security properties for an object (Figure 16-6) can change, so can the permissions listed in the access rights box, depending on the object(s) to which permissions are being applied. A good demonstration of this is to open up the security properties for any user and scroll through the displayed list of permissions. Next, open up the wizard on any container and select "Create a custom task to delegate" (see the screen shown in Figure 16-11) and only user objects (see Figure 16-12). The screen shown in Figure 16-13 should then display the same list of permissions that the screen in Figure 16-6 does. This makes sense—available permissions for one user should be the same as the available permissions for all users—but is still nice to see the correlation in the flesh, so to speak.

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 16-6 and click on the Auditing tab, a screen similar to that in Figure 16-14 is displayed.

This window shows the list of auditing entries (AEs) that have been defined on the object. It's not very helpful viewing them from here, though, as the detail is limited. So, just as you would do with permissions, you can select an entry, click the Edit button, drill down, and view the individual AE itself.

Figure 16-15 shows the successful and failed items that are being audited.

Designing Permissions 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 permissions 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.

The Five Golden Rules of Permissions Design

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

The rules are:

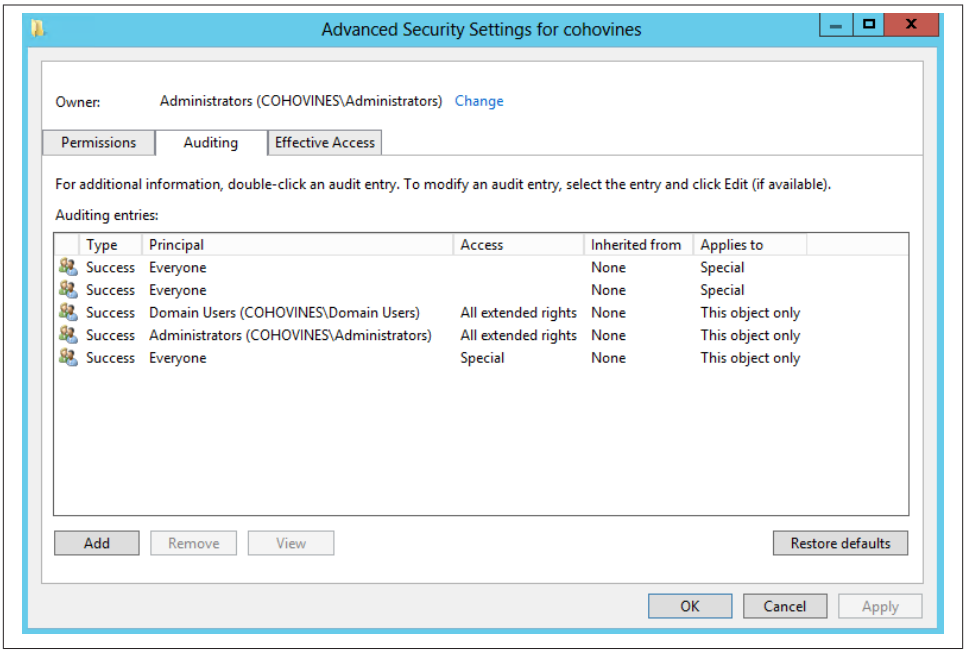


Figure 16-14. Advanced Security Settings window showing auditing entries

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

Let's look at these rules in more detail.

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.

We're not advocating the use of one group or two, as we'll explain in more detail in the next section on how to plan permissions. We are advocating that whichever way you choose to implement group permissions, you should add users to groups and apply

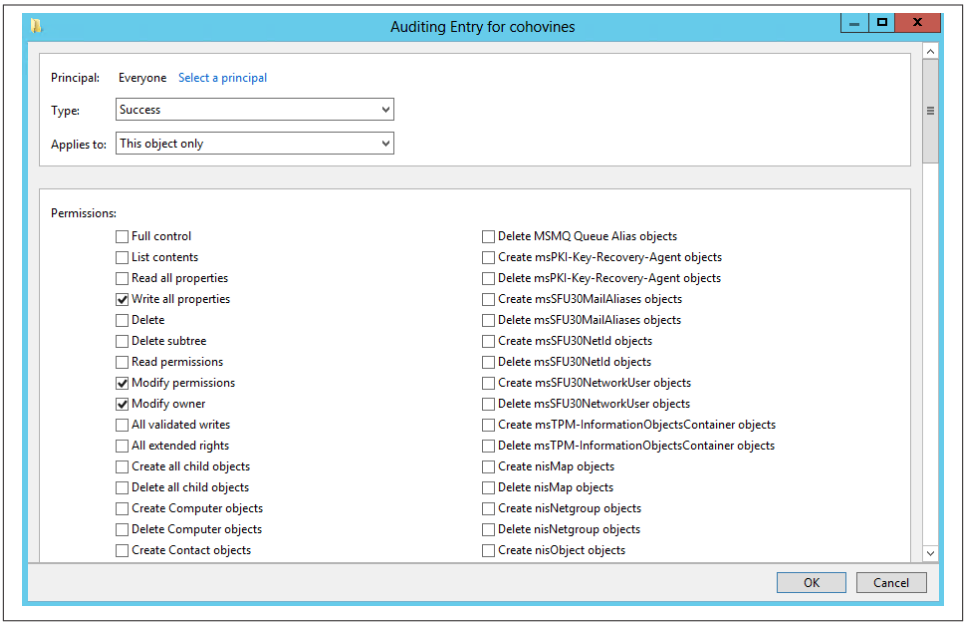


Figure 16-15. Auditing entry for an object's properties

permissions to groups, even if the group initially contains only one user. This removes organizational dependence on one particular account.

Time after time, we have seen organizations in which individual users with a whole raft of permissions to various objects suddenly leave or change roles. The new administrator then has to go in and unravel differing permissions settings from this account. We have even seen one administrator, looking in anguish at the tangled mess of a recently departed administrator's account, delete his own account and rename the departed user's account just so that he could get the correct permission set without having to figure out the mess! If the old administrator had been a member of, say, five different groups, each with the relevant permissions, the new administrator could simply have replaced the group memberships of the old account with his new account. This is a much simpler approach, and we are sure that none of the preceding common sense is very new to system administrators.

Rule 2: Design group permissions so that you have minimal duplication

It makes much more sense to create groups with simple, distinct permission sets than it does to create multiple groups with overlapping permissions. If you decide that some of your users need, say, create and modify permissions, while others need modify and delete permissions, and a third set of users need just modify permissions, it makes much more sense to create three separate groups with create, delete, and modify rights,

respectively, than it does to make three groups with the permissions sets described. Let's consider an example. Suppose we were to create three groups called *Create and Modify*, *Modify and Delete*, and *Modify*. Now say we added 10 users to each group. If the only modifications ever to happen to these groups were occasional membership changes, this solution would fit adequately. However, suppose that (as happens in every large organization) the permissions requirements were to change over time. If Dave Peace, a member of *Create and Modify*, now needs delete permissions, what do we do? Do we make a special case of Dave's account and add the delete permission to his account only? Arguably, that is the simplest solution, but according to Rule 1, we really should create a group to manage the permission set. Do we create a *Delete* group and add Dave's account to it, or create a *Create, Modify, and Delete* group and move his membership from *Create and Modify* to the new group? Both are valid solutions.

Let's say we go with the former and create a *Delete* group, adding Dave as a member of that group. Then things change again, and Mark Newell joins the organization. Mark needs all three permissions, so we add him to *Create and Modify*—but do we also add him to *Modify and Delete* or just *Delete*? Either way, we now have potential confusion. Whenever we have to check for members who can modify and delete, we have to check three groups, not one.

If we'd taken the second approach and chosen to create *Create, Modify, and Delete* rather than the *Delete* group, we could simply add Mark to *Modify and Delete* when he joins, and things would seem to work fine. Now suppose Paul Burke moves from another team and requires create permissions only, so a *Create* group is created and his account is added to that. Later, three others join the *Create* group, but Paul now needs create and delete, so *Create and Delete* is created, and he is moved to this group. Now we have six groups: *Create*, *Modify*, *Create and Delete*, *Create and Modify*, *Modify and Delete*, and *Create, Modify, and Delete*. Unfortunately, if we ever have to check who has create and modify permissions, we have to check three groups: *Create*, *Modify*, and *Create and Modify*.

The minimal solution that would require the fewest ACEs and groups would be to have one group for each permission granted, *Create*, *Modify*, and *Delete*. Users can then be added to the groups as needed. If you prefer a more "role"-based approach, you could create role-based groups such as *Create and Modify* and the others and nest them in the *Create* and *Modify* groups that have the actual permissions in the directory.

This example was heavily contrived. However, we hope it serves to show that duplication will occur whenever you have users requiring separate permissions to an object or property and users requiring combinations of those permissions. It is for this very reason that we suggest creating separate groups for the lowest-common-denominator permissions that you need to set. Keep possible future enhancements and role redefinitions in mind, because you won't usually want to be changing permissions in the directory in an ad hoc manner any time new requirements pop up.

If you have users who always need read, list, and create permissions but require different combinations of delete and modify, it may not make sense to have the three groups—one each for read, list, and create. You could instead create one group with the read, list, and create permissions assigned to it, one group for delete, and one for modify. Then you could use multiple group memberships to assemble the group permissions as you require them. Of course, if later you have a requirement to grant only read or create permissions, you'll end up separating out the permissions anyway. Try to think ahead.

The most important point to note is that we are talking about minimizing and simplifying the number of ACEs applied to the directory. If you need to grant only create, modify, and delete rights to an object across the board, you probably don't need to create three groups. But don't necessarily rule out this approach if you think the requirements are apt to change.

If, after you have created a group with multiple permissions, you find that you now need groups with individual permissions, you can always create the smaller groups and migrate the users. Active Directory is flexible enough to allow you to operate like this, but it can be considerable work to clean up after the fact and must be done in a slow, painstaking way to avoid impacting the users.

Rule 3: Manage advanced permissions only when absolutely necessary

(Please note that this says "permissions" and not "auditing." Auditing entries can be accessed only from the Advanced tab, so this rule makes less sense for auditing entries.)

Whenever you right-click an object to view its properties, the security properties window that appears has an Advanced button on it. This was shown in Figure 16-6 in the section "Using the GUI to Examine Permissions" (page 451). The security properties window itself typically has the following allow and deny options as general permissions:

- Full Control
- Read
- Write
- Create All Child Objects
- Delete All Child Objects

The general permissions are not limited to those five, though, and indeed they change depending on the object you are looking at. For example, the security properties for any user object display additional general permissions, such as Reset Password, Modify Web Information, and Send As. While these general permissions make sense for the user object, they are not all appropriate for other objects. This rule suggests that you manage permissions for objects from the security properties window as often as you can. Use the Advanced button only when you wish to allow or deny a permission to one aspect of an object rather than the whole object. An example would be manipulating the

permission to read or modify a user object's telephone number rather than the account details as a whole.

While there is nothing wrong with managing atomic permissions to objects and properties, permissions are much easier to manage from a higher level. The main permissions that administrators might want to set were put here for this express purpose, so that users and groups can easily manage the tree without having to worry about the large amount of individual properties.

If you choose to get very granular with permissions, you will want to look at using *DSACLS* at the command line for setting the permissions. *DSACLS* does a better job of showing all of the permissions and in some situations is more intelligent about ACEs that are applied than ADUC. Finally, *DSACLS* can be used in scripts for consistent results.

Rule 4: Allow inheritance; do not protect sections of the domain tree from inheritance

If you allow or deny permission for a group or user to all objects of a certain type in a container, by default the permissions are applied recursively to all objects of that type in all the child containers down the tree. It is possible to block inheritance, but we recommend leaving inheritance in place (the default) and protecting branches on an individual basis only when there are good justifications for doing so. The reason is simple: if you specify that children do not inherit permissions from their parents, you are adding additional complexity to your Active Directory environment. There are several examples of inheritance blocking found in every default domain. You can simply look at any administrative ID, or alternatively drill down into the System container, then the Policies container, and look at any of the `groupPolicyContainer` objects.

The `groupPolicyContainer` objects are protected for similar security reasons. As discussed in Chapter 11, these objects contain information about security policies applied to the domain. Microsoft rightfully decided to protect these objects from being accidentally impacted by permissions delegations higher up in the directory. You will not generally have to manipulate these objects directly, so you could manage Active Directory for years and never notice these objects; the complexity involved with protecting them won't directly impact you.

Ultimately, there is nothing wrong with protecting objects or sections of the tree from inheritance. However, it is important to remember that every time you do it, you are possibly creating more work and potential confusion for yourself and other administrators. As an administrator, you should keep track of these changes in a log, so that you can easily reference your special cases when required.

Rule 5: Keep a log of changes

This may sound like an obvious statement, but it is surprising how many administrators overlook such a simple requirement. Simply put, it is always wise to keep a log of custom

changes that you make to a default installation so that you and others have something to refer back to. There will be times when you may not be available and this sort of information is required. The following list shows the relevant fields of a basic Active Directory ACL log:

- Unique name of object or LDAP location of object in tree
- Object class being modified
- Object or property being modified
- User or group to whom permissions are being assigned
- Permissions being assigned
- Notes on reasons why this change is being made

Some additional items outside of ACL changes worth logging are schema default security descriptor changes, property set modifications, attribute index changes, and attribute confidentiality changes.

Let's now look at how you can put these rules into practice in your own designs.

How to Plan Permissions

There are a number of Active Directory Users and Computers permission sets that administrators may need to implement in their organizations. Some examples are:

- A set of centralized teams, each with responsibility for certain areas. Users can be members of more than one team: account modifiers, printer managers, computer account managers, publishing managers, and so on.
- A manager for each individual major organizational unit under a domain.
- A manager for each individual major organizational unit under a domain who is able to delegate responsibility for organizational units lower down the tree.

While we could go through each of these cases and show how to design permissions for each, every organization is different. For that reason, it seems better to try to show what we consider to be the best method to use when designing Active Directory permissions for all types of organizations.

First, create two documents, one called *Allow* and the other called *Deny*. On each document, label two sections, one called Global Tree Permissions and the other Specific Tree Permissions. Place two subheadings under each of the two sections, calling one General Permissions and the other Special Permissions. Then, under each general and special heading, create three columns: "LDAP path," "What to set," and "To whom."

The first six columns relate to permissions that will apply throughout the whole tree; the last six relate to permissions that will apply to specific locations in the tree. The latter

is likely to be the much larger of the two lists. The General columns relate to permissions that can be set without needing to use the Advanced button, such as read access to all objects below an organizational unit. The Special columns relate to those permissions that you have to manually bring up a permissions editor window for, such as allowing read access to all telephone numbers of user objects below a particular organizational unit. In each section, the "LDAP path" column should display the LDAP path to the object that is to have properties set, the "What to set" column should show the permissions that are being set, and the "To whom" column should display the group or user to whom the permissions are being assigned.

The LDAP path under Global Tree Permissions is, strictly speaking, unnecessary, since these columns relate to permissions applied to the domain as a whole. If, however, you have a special need to apply permissions to a large number of organizational units directly below the root, you could use this column for this purpose.

Now go through your Active Directory design and begin to populate both the Allow and Deny tables. For a first pass, you should concentrate on a thorough design, listing all the permissions that you think you will need. Print out a number of copies of the table. Once you have a list in front of you, you can start amalgamating identical permissions into groups. It is likely that you will need to go through many iterations of these designs until you get a pared-down list that you are happy with. As you go through the design, you will start identifying groups of users to which you need to apply permissions. When designing the groups, you have two choices, as previously discussed under Rule 2. You can either create a single group to which permissions are applied and that contains users, or you can create two groups: one to apply permissions to and one to hold users, which is nested in the first group.

The decision on whether to go for single or dual groups is not necessarily an easy one. Our preference is to use single groups as often as possible, unless we need extra flexibility or have a lot of permissions to assign to many groups. In order to help you to make a bit more sense of the decision, a few reasons why you would want to consider one or the other are shown in Table 16-3.

Table 16-3. When to consider user groups and permission groups or combined groups

You should consider one group if	You should consider two groups if
You want to keep the number of groups to a minimum.	You want greater flexibility. Having one group for applying permissions and one for users means that you are always able to manage the system with a minimum of fuss.
You have only a small or simple tree, where it would be fairly easy to track down problems.	You have a large or complex tree, in which you need to be able to identify any problems quickly.
You need to assign only a few simple permissions.	You need to assign a large number of permissions.
You have very little change in the membership of groups and very few changes to permissions.	You have regular changes in your group membership or permissions.

You should consider one group if	You should consider two groups if
You have little cross-membership of groups.	You have major cross-membership of groups, where a user could exist in more than one group with conflicting permissions. (Two groups make it easier to debug problems in a large environment.)
You very rarely need new groups.	You regularly need new groups with subsets of your existing users who have been assigned to some task.
You very rarely have to split user groups so that each user group subset has different permissions than the original group had.	You regularly have to split an existing group into more than one group, because each requires a different set of permissions than the old group used to have.

One last point: if you are creating permission groups and user groups, remember to name them sensibly from the outset. Maybe use something such as *Finance – Allow User Modify* for a group granting user modification rights on the Finance OU and *Finance – User Admins* for the user admin role over the Finance OU. It makes it easier when managing and scripting if you can easily identify which types of groups are which.

Bringing Order out of Chaos

We've had people ask what we would recommend to someone arriving at a new company where the previous directory services administrator had left a tree with no proper permissions design or consistency. In this situation, start by analyzing what the organization requires and work from there. You also should analyze what permissions Active Directory currently has assigned, although concentrating solely on this could be detrimental. After all, if the last administrator was not competent or knowledgeable enough to set up a sensible permissions scheme from the start, he may not have accurately implemented the permissions required by the organization.

When analyzing Active Directory, you need to start by identifying the members of the built-in groups on the server, such as Domain Admins, Backup Operators, and so on. Now do the same for the other groups that are specific to the organization. Once this is done, using the previously described tables, you need to list the permissions on the root of the first domain in the tree you are responsible for. Then look at the permissions for the first container or organizational unit in that list. Next, navigate through the branch represented by that container, looking sequentially at the child containers, continually recursing through the tree. Once this first branch of the root is mapped out for the container permissions, you should be getting an idea of what permissions are required. Now go through all the objects in that branch, including printers, users, shares, and so on. This is time-consuming and annoying, but after a while you should start getting an idea of what is going on. All of this is just a sensible approach to going through Active Directory, not a quick-fix solution. You still have to continue throughout the domains you are responsible for to complete the process. It is also often helpful to use a script to iterate through Active Directory and display all of the ACLs for an object.

Your first main goal should be to move the individual user permissions to groups with users assigned to them as often as possible, thus making Active Directory simpler to manage and comprehend. These groups should be sensibly named for what they do rather than whom they contain (after all, you are looking to understand Active Directory first). Ideally, you can start consolidating users with identical permissions into the same group.

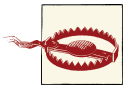
Your second goal is to remove permissions that users or your newly created groups should not have. This may, of course, mean that your new groups need to have their members split into two or more separate extra groups. For example, a group that has Read All Properties rights and Write All Properties rights to an object may need to be split into three groups with more limited permissions instead: one with Read All Properties, one with Write All Properties, and one with selected write permissions rather than complete write access. This may be evident from your Active Directory analysis, or it may come out of discussions with users or their managers, with whom you should at least confirm the changes before implementing them just to make sure your analysis is correct.

Ultimately, your third goal, having rationalized the array of Active Directory permissions, is to try to limit the inheritance blocking of objects and branches and to try to move as many advanced permissions to general permissions as you can. You might think that it makes more sense to do this earlier, and in some cases, this is true. However, if you complete the first two goals, you will have an Active Directory tree that you understand and that has been brought back into line with sensible rules. It is much easier to attempt to fix problems with inheritance blocking and advanced permissions once you have a manageable and rationalized tree. You may end up going back and changing groups or permissions that you already sorted out in attaining the first two goals, but consider how much more difficult it would be to attempt to do these concurrently. After all, you are trying to make the best of an already annoying task. There is no sense in trying to do everything at once. As you go through the tree checking for inheritance blocking, you should document the blocked objects and branches, as specified in Rule 5, just as if you had set up the blocking from scratch yourself. That way, you can use the tables to analyze and keep track, crossing off those that are of no use as you rationalize the tree.

This whole section can be boiled down to two simple words: simplify and secure. You want to end up with the simplest, least delegated model you can attain while meeting the requirements at hand, resulting in the most locked-down directory you can live with.

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 any object in the Domain naming context (NC) will propagate domain-wide 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, because changes can occur on any DC, you will need to collect and combine 100 security event logs to gather a complete picture.

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

- Someone complains that user details are being modified without a known cause.
- You notice that objects have been unexpectedly created or deleted in a container.
- The Active Directory hierarchy has changed and you weren't informed.
- You suspect a security problem.

Although the preceding reasons are all great reasons for enabling auditing, they are better reasons for removing native update access rights to the directory and pushing updates through some sort of provisioning system, such as Microsoft's Forefront Identity Manager product (FIM, formerly known as ILM and MIIS), Quest's Active Roles Directory Management tool, or even home-grown web-based tools. With these kinds of tools, you can implement good, solid features that are sorely missing from Active Directory, such as the following:

Centralized logging
: Enables all information concerning all requested changes to be easily deposited in one location with no additional work.

Business rules
: Allow you to force naming standards or specifically allow/disallow values in attributes.

Triggers
: Send notifications to various individuals when certain things are requested, such as notifying the asset management group of a new computer being added to Active Directory.

Custom authorization levels
: Allow you to set limits; for example, to specify that a data admin can create no more than five users a day.

Prevention rather than cure
: Auditing the directory for things that shouldn't have happened is reactive; securing the directory so those things can't happen in the first place is proactive and involves considerably less work in the long run.

In all of 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. Once you have written up each scenario, if 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.

Implementing Auditing

Windows Server 2008 introduced a completely new Active Directory auditing capability that greatly improves upon the capabilities in Windows Server 2003. On Windows Server 2003 domain controllers, if you enabled the "Audit directory service access" setting, a generic event was logged each time a user succeeded or failed to access an object with a System access control list (SACL) defined. From a practicality standpoint, this functionality was of limited value and could easily flood the security logs on your domain controllers.

Active Directory now allows you to audit changes to the directory on a granular level, as well as to view previous and current values when changes occur. Figure 16-16 shows an example of an attribute-modify audit event. A number of new auditing events have been created that will be logged to the security log when changes occur. See Table 16-4 for a listing of these events.

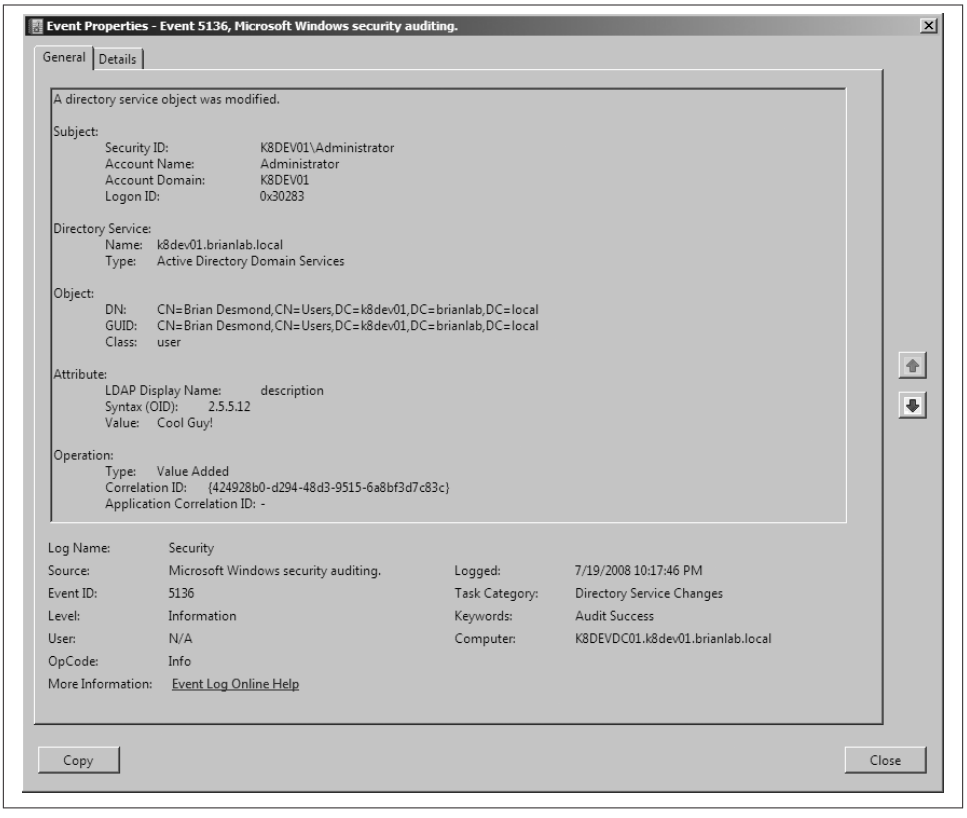


Figure 16-16. Attribute change audit

Table 16-4. Windows Server 2008 and newer auditing events

Event ID	Change type	Description
5136	Modify	Logged anytime an attribute on an audited object is changed when the relevant Write Property audit is set in the SACL
5137	Create	Logged anytime an object is created under a parent when the Create Child audit is set in the SACL
5138	Undelete	Logged anytime an object is reanimated under a parent when the Create Child audit is set in the SACL
5139	Move	Logged anytime an object is moved to a parent when the Create Child audit is set in the SACL
5141	Delete	Logged anytime an object is deleted when the Delete Self audit is set in the SACL, or when an object is deleted under a parent when the Delete Child audit is set in the SACL

The Directory Service Access event 566 that was logged under Windows Server 2003 and Windows 2000 is still present; however, the event ID has changed to 4662.

Enabling the new auditing functionality is a three- or four-step process:

1. Enable the "Audit directory service access" Group Policy setting.
2. Configure the "Advanced audit policies" Group Policy settings.
3. Configure the SACL on the objects in the domain that you want to audit.
4. Exclude attributes from auditing forest-wide by modifying the attributes definitions in the schema.

To enable the "Audit directory service access" setting, browse to it under *Computer Configuration\Policies\Windows Settings\Security Setting\Local Policies\Audit Policy* inside of a group policy that applies to your domain controllers, and audit *Success* attempts.

Next, you'll need to configure "Advanced audit policies" in group policy. These settings can be found under *Computer Configuration\Policies\Windows Settings\Security Setting\Advanced Audit Policy Configuration\Audit Policies\DS Access*.

In order for change audits to be logged, you will need to configure the SACL on the objects to be audited, as discussed earlier in this chapter. Make sure to enable the correct auditing entries, as outlined in Figure 16-15.

Change auditing can be disabled globally for attributes by modifying the `searchFlags` attribute on the `attributeSchema` object in the Active Directory schema. The `searchFlags` attribute is a bit mask, and auditing is managed by the ninth bit. To disable auditing for the `info` attribute using *adfind* and *admod*, you would run the following command:

```
adfind -sc s:info systemFlags -adcsv | admod systemFlags::{{.:SET:256}} -upto 1
```

To enable auditing for the `info` attribute, run this command:

```
adfind -sc s:info systemFlags -adcsv | admod systemFlags::{{.:CLR:256}} -upto 1
```

You must be a member of Schema Admins to modify this attribute.

Tracking Last Interactive Logon Information

Active Directory supports a series of attributes that, when enabled, store the date and time of the last successful and failed Ctrl-Alt-Delete logons for a user in the directory, as well as the hostnames of the machines on which they occurred. These attributes are part of the `user` class in the directory. You may be wondering how this is different from

the `lastLogonTimeStamp` attribute. The `lastLogonTimeStamp` attribute uses special functionality to limit the update frequency to an infrequent (every 10 to 14 days, roughly) interval, and it does not store the hostname associated with the logon attempt. These new attributes are updated immediately.

There are a number of limitations to this feature that you should keep in mind when considering whether or not it is appropriate for your environment. The first is that this functionality only tracks interactive logons. In other words, only logons that occur when a user presses Ctrl-Alt-Delete on a Windows Vista or newer machine are tracked. In order for the tracking to occur at all, the user's domain must be running at the Windows Server 2008 functional level or better, and a registry setting must be enabled on the client and on the domain controllers.

The registry setting is a `REG_DWORD` called `DisplayLastLogonInfo` located at `HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System`. If, on a domain controller, you set `DisplayLastLogonInfo` to `1`, it will begin storing logon statistics in Active Directory. If you set this value to `1` on a client, the client will begin reporting last logon statistics similar to what's shown in Figure 16-17 when a user logs in.

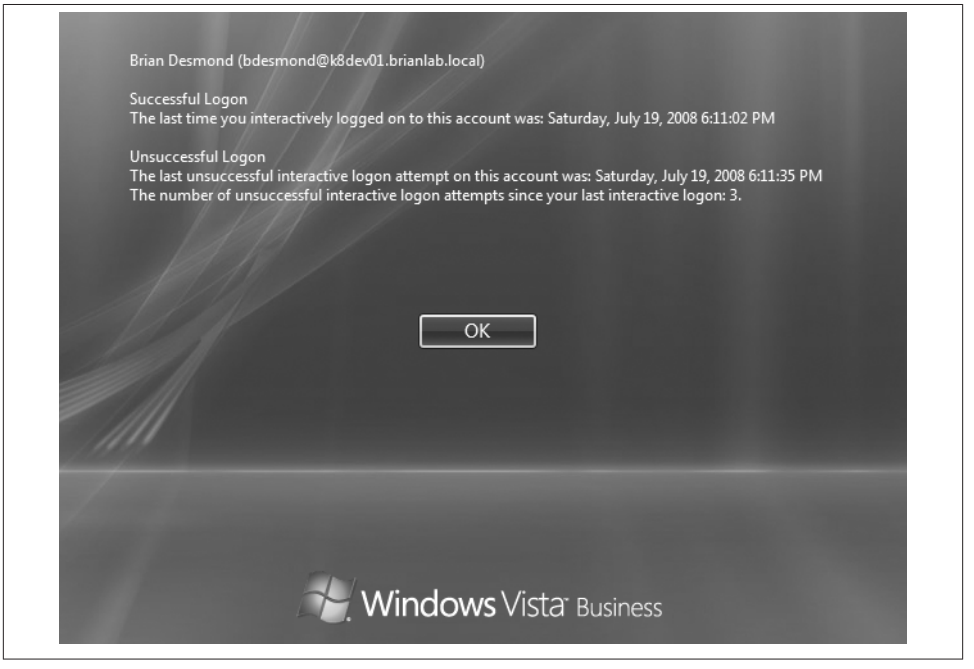


Figure 16-17. Displaying last interactive logon statistics

In lieu of setting a registry setting, we recommend that you instead use the corresponding Group Policy setting for this feature. The Group Policy setting is available under

Computer Configuration\Policies\Administrative Templates\Windows Components \Windows Logon Options. The particular setting you want to enable is "Display information about previous logons during user logon." This setting needs to be enabled in policies that apply to both domain controllers and clients in order to enable storage and display of logon statistics, respectively.

If you enable this setting on clients that have users logging into domains not running at the Windows Server 2008 or better functional level, clients will receive an error similar to Figure 16-18 and be unable to log in. You will also receive the error in Figure 16-18 if this setting is enabled on clients, but not domain controllers.

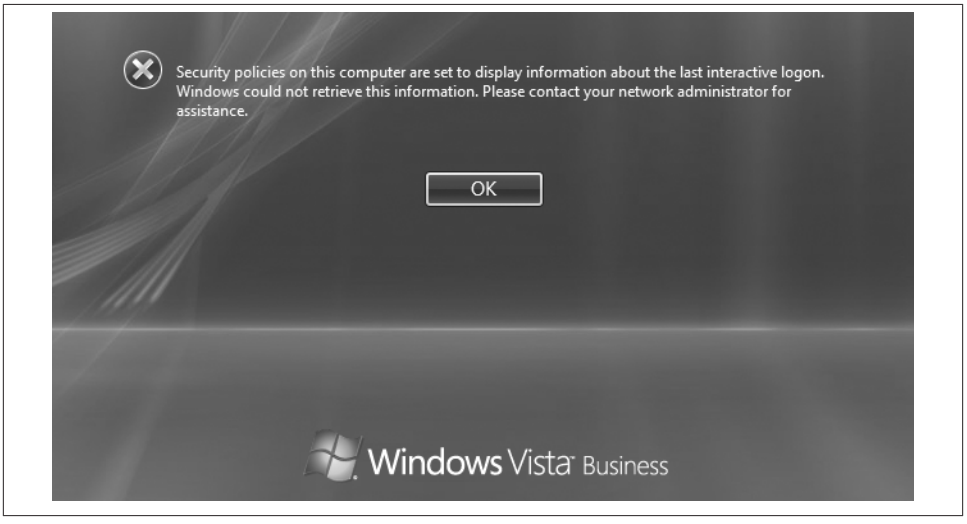


Figure 16-18. Last interactive logon statistics failure

There are four attributes that are used to provide this additional logon information functionality in the directory:

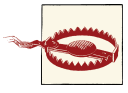
`msDS-LastSuccessfulInteractiveLogonTime`
: This is the time of the user's last successful Ctrl-Alt-Delete logon.

`msDS-LastFailedInteractiveLogonTime`
: This is the time of the user's last failed Ctrl-Alt-Delete logon.

`msDS-FailedInteractiveLogonCount`
: This is the number of logon failures that have occurred for the user since the feature was enabled.

`msDS-FailedInteractiveLogonCount AtLastSuccessfulLogon`
: This is the number of failed Ctrl-Alt-Delete logons for the user since the last successful Ctrl-Alt-Delete logon.

Unlike the `lastLogonTimeStamp` attribute, there is no update frequency safeguard for these attributes. If you enable this feature, keep in mind that every single time these attributes are updated, they will replicate in the next cycle. There could potentially be thousands of updates a day to these attributes in a busy domain. Also, there is no guarantee that the values shown to the user (like in Figure 16-17) are in fact the most accurate values in an environment with more than one domain controller in the domain, since these attributes replicate like any other change to the directory.



If you have deployed read-only domain controllers, they will update these attributes locally, but the changes will not be replicated to the rest of the domain.

The implementation of the last interactive logon statistics feature discussed in this section was designed primarily for compliance with Common Criteria (*http://www.commoncriteriaportal.org*). Unless you have a business need for this feature or you are enabling it in a small (e.g., single-site) domain, we do not recommend that you enable it.

Real-World Active Directory Delegation Examples

It now seems appropriate to put what we have laid out in this chapter 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.

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 the engineer to get an arrival-time update rather than ringing the central desk. Management has decided that they want to restrict who can ring the pagers or mobiles, but they do not want to stop giving out the engineers' names 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 `pager` and `mobile` properties of the engineer objects by default. The first thing we need to find out is how the permissions are currently granted to the users. We look at the ACL on a user object and we do not see either attribute listed specifically. We do, however, see an inherited Read Property (RP) access for the entire object granted to the Pre-Windows 2000 Compatible Access group. We also see explicit RP access for several property sets granted to Authenticated Users.

Next, we look at the schema definition for the two attributes and we see that both have the `attributeSecurityGUID` set to `{77B5B886-944A-11D1-AEBD-0000F80367C1}`, which means the attributes are in a property set. We scan through the `cn=extended-rights` container and determine that the property set these attributes are in is the Personal Information property set. We look at the ACL again and see that the Personal Information property set is one of the property sets that has RP access granted to Authenticated Users. We know that the domain has Pre-Windows 2000 Compatible Access enabled, so users are getting access to the `pager` and `mobile` attributes through two different ACEs, both explicit and inherited. Recall from the earlier section "Permission Lockdown" on page 433 that the explicit property set ACE is the more difficult of the two to deal with. We will focus on that one initially.

The first thing we will do is create a sub-OU under Hardware Support Staff called Engineers and move all of the engineer user objects into that OU. We could try to individually lock down the engineer user objects in the parent OU, but that is a layer of complexity that isn't needed and should be avoided. Now that all of the engineers are isolated in their own OU, we need to decide which of the previously discussed mechanisms is appropriate to lock down access to the `pager` and `mobile` attributes. Again, these options are:

1. Remove the `pager` and `mobile` attributes from the Personal Information property set.
2. Change the `defaultSD` of the user object class to remove the Personal Information property set.
3. Strip the Personal Information ACE from each of the engineer objects and add inherited deny ACEs for the `mobile` and `pager` attributes.
4. Add an explicit deny ACE for the `mobile` and `pager` attributes on each engineer object.

Options 1 and 2 could be used, but we would have to go back through the rest of the forest and add inherited ACEs granting RP access to the `pager` and `mobile` attributes for Authenticated Users. All of this extra work would be needed to attain the net effect of a permission change only on the targeted OU. If you think that you will be locking these attributes down for more OUs or maybe for all users in the forest, these options would become more palatable.

With option 3, you target only the engineer objects. When you strip the Personal Information ACE, not only will access to the `mobile` and `pager` attributes be affected, but also access to every other attribute in the Personal Information property set, which includes 46 attributes in the default Windows Server 2008 forest.

For all three of these options, if the domain is in Pre-Windows 2000 Compatible mode, users will still have access to all attributes through the inherited ACE granted to the Pre-Windows 2000 Compatible Access group. In order to lock down the two targeted attributes, we would also need to add an inherited deny ACE for each attribute on the Engineers OU. This would work great and everyone would be happy until later on, maybe a year or two down the road after you've forgotten all about this change, when you decide, "We don't need to be in Pre-Windows 2000 Compatible mode anymore" and remove the Authenticated Users group from the Pre-Windows 2000 Compatible Access group. As soon as you do this, the permissions removed in options 1, 2, and 3 would quite suddenly impact you by disallowing access to all attributes that are in the Personal Information property set. Obviously, this could be quite a surprise! You might think it was because you still needed to be in Pre-Windows 2000 Compatible mode, but it would actually just be a delegation issue.

The last option, option 4, would target just the engineer user objects, and there would be no unintended side effects later with other domain changes. This is probably the best solution in this case, though it will require is a good amount of initial work plus ongoing maintenance as new engineer users are added. It would be best if a script were built strictly for doing this work.

You may wonder, "Wouldn't this be a great place to use the confidential bit?" Unfortunately, no, this isn't a good example, because it would impact all users in the forest; also more importantly, both the `mobile` and `pager` attributes are Category 1 attributes, so you can't modify them to make them confidential.

Allowing Only a Specific Group of Users to Access a New Published Resource

The Finance Department has created a new share called *Budget_Details* and published it in the tree. The permissions to the share itself allow only the correctly authorized staff to read and view the contents, but everyone by default can see the volume object describing the share in the tree. The Finance Department does not want the share visible in Active Directory to anyone who is not a member of the Finance group.

Again, the first thing we need to determine is how the permissions are currently granted to the users. We look at the ACL on a volume object, and the only permission granted to normal users is a Generic Read ACE for Authenticated Users. Since this is a single object, it probably doesn't make sense to create a whole OU for it, so we open up the permissions editor window for the volume object and remove the Authenticated Users

group entry. We then add an entry for the Finance group and assign Read and List permissions.

Restricting Everyone but HR from Viewing National/Regional ID Numbers with the Confidential Bit

After much discussion, it is decided that national/regional ID numbers will be moved from a proprietary database and inserted into Active Directory. There is quite a bit of concern about the visibility of the attribute, as only the HR Department and Domain Admins should be able to see it by default.

You extend the schema with a new national/regional ID number attribute and you set the `searchFlags` attribute to `128` to indicate that the attribute should be protected using the confidential attribute feature. Now you simply have to create a new group called *HR-Confidential* and assign a single inheritable ACE at the root of each domain that grants this group Control Access to the national/regional ID number attribute on user objects.

Later, if someone decides they want to secure the attribute from domain administrators as well, you will need to go through and apply an explicit deny ACE to every single object for that group. However, trying to deny access to data in the directory to domain administrators is not actually enforceable. At any point that a domain admin would like to get access to the attribute again, she can simply modify the ACL herself. If you absolutely cannot allow domain administrators access to the data, you need to find some other way of protecting the data, such as using third-party encryption or placing it on a machine that isn't in the domain and therefore is not under the control of the domain administrators.

The AdminSDHolder Process

One of the background processes in Active Directory that has been a source of confusion for administrators since Windows 2000 is *AdminSDHolder*. This name derives from the name of the object in Active Directory that stores the security descriptor that is applied in order to protect sensitive objects. Sensitive objects are users, groups, and computers that are members of the following groups::

- Account Operators
- Administrators
- Backup Operators
- Domain Admins
- Domain Controllers
- Enterprise Admins

- Print Operators
- Read-Only Domain Controllers
- Replicator
- Server Operators
- Schema Admins

In addition, the following objects are explicitly protected, irrespective of group membership:

- Administrator
- krbtgt

Active Directory protects these objects in order to prevent privilege elevation by users who have been delegated rights over members of these groups. Typically these are users who are delegated administrators, e.g., in the helpdesk. If a helpdesk user was allowed to reset the password of a member of the Domain Admins or Account Operators group, for example, he could reset the user's password and then log in as that user—greatly elevating the helpdesk administrator's privileges.

The *AdminSDHolder* process runs in the background on the PDC emulator FSMO role holder every 60 minutes by default and applies a special ACL that does not inherit permissions to each object that is in scope. The ACL is stored in the `ntSecurityDescriptor` attribute of the `CN=AdminSDHolder,CN=System` object under the root of the domain.

If you want to modify the frequency at which the *AdminSDHolder* process runs, you can set the `AdminSDProtectFrequency` registry setting (`REG_DWORD`) located in `HKLM\SYSTEM\CurrentControlSet\Services\NTDS\Parameters` to a value between `60` and `7200` (seconds). We recommend leaving the default, `3600`, in place.

Administrators often run into this process for the first time in a couple of scenarios. The first is in the case that a user used to be an administrator but then had those permissions revoked. The second common scenario is with Exchange 2010 and newer. If users are using their day-to-day user accounts for administrative tasks (not recommended), the first time they try to connect a mobile device for ActiveSync, the connection will fail as Exchange needs additional access to create a special object under the user's account to track the mobile device.

There is no automatic "undo" process for removing the protection *AdminSDHolder* adds when an object is no longer considered to be protected. Instead, you must manually

configure the object to once again inherit permissions. In addition, you should reset the `adminCount` attribute's value to `0`.

Both of the scenarios discussed here can be avoided by implementing the best practice of using a second user account for all administrative tasks and treating an administrator's standard account as a least-privileged account that is identical to the accounts for traditional employees. You could also modify the *AdminSDHolder* object's ACL, but this would simply avoid confronting poor security design.

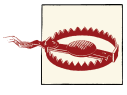
In some cases you may want to exclude some of the groups listed earlier from *AdminSDHolder*. Active Directory allows you to exclude the following groups (and their members) from protection. In general, we *do not* recommend that you use these groups, but if you choose to and have a need to exclude their members from protection, you may. The groups are:

- Account Operators
- Server Operators
- Print Operators
- Backup Operators

If a user who is a member of one of these groups is also a member of another protected group, that user will continue to be protected by *AdminSDHolder*.

To exclude any of these groups, you will need to modify the `dsHeuristics` attribute with ADSI Edit by following these steps:

1. Launch ADSI Edit (Start→Run→**`adsiedit.msc`**) and open the Configuration NC.
2. Browse to `CN=Windows NT,CN=Services`.
3. Open the properties of the `CN=Directory Service` object.
4. If the `dsHeuristics` attribute is not set, set it to `000000000100000f`. If the attribute is already set, move on to the next step.



Do not overwrite the `dsHeuristics` attribute with the default in step 4 if there is already a value set. There are many other behaviors that are controlled by `dsHeuristics`!

5. Each of the first four characters in the `dsHeuristics` attribute controls excluding one of the four groups listed earlier:

a. To exclude Account Operators, set the first character of `dsHeuristics` to `1` (e.g., `100000000100000f`).

b. To exclude Server Operators, set the second character of `dsHeuristics` to `1` (e.g., `010000000100000f`).

c. To exclude Print Operators, set the third character of `dsHeuristics` to `1` (e.g., `001000000100000f`).

d. To exclude Backup Operators, set the fourth character of `dsHeuristics` to `1` (e.g., `000100000100000f`).

Once you have completed your changes, you may want to force the *AdminSDHolder* process to run so that you can test your changes. You can do this by invoking the background process with an *operational attribute* using LDP:

1. Launch LDP and then bind to the PDC emulator FSMO role holder.
2. Open the Browse→Modify dialog.
3. Enter `RunProtectAdminGroupsTask` in the Attribute box, as shown in Figure 16-19.
4. Enter `1` in the Values box and then click Enter.
5. Change the operation to Replace.
6. Click Run.

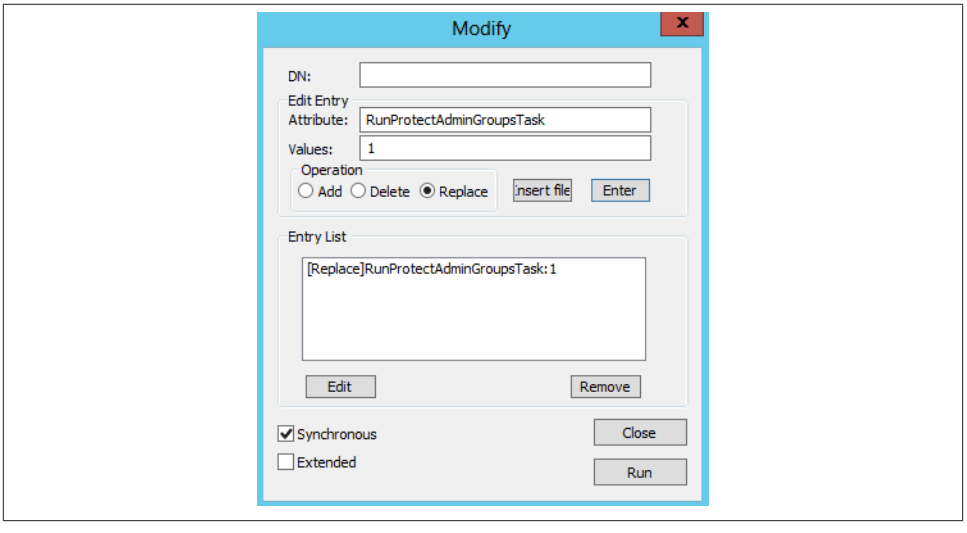


Figure 16-19. LDP attribute Modify dialog

Dynamic Access Control

One of the most talked about new features in Windows Server 2012 is Dynamic Access Control. DAC attempts to solve a problem that has plagued Windows administrators forever—managing filesystem access control lists and ever-changing group memberships. One of the challenges with the NTFS ACL structure is that you cannot construct a compound expression. That is, you can't grant read access to a file or folder for users who are members of both the Finance Users group *and* the Managers group. In order to meet this requirement, you need to create a new group—say, Finance Managers—and grant that group read access. Over time, the number of permutations that arise due to common scenarios like this leads to an explosion of groups in Active Directory, token bloat issues, and IT management challenges related to keeping all these groups current.

DAC aims to solve both of these problems by moving to an expression-based ACL structure for the filesystem that is based on both group membership and attributes of the user object in AD. These attributes are expressed as *claims* and are added to the user's security token when the user is authenticated by a Windows Server 2012 DC.

If a user is authenticated by a DC running a version of Windows prior to Windows Server 2012, a Windows Server 2012 file server can reach out to a Windows Server 2012 DC in the file server's site and obtain an access token that includes claims data.

In addition to attributes of the user, Windows 8 workstations can participate in DAC through a feature called *compound identity*. Compound identity enables claims from the user's workstation to be included in the user's token, and thus the access control expressions on the resource. With this functionality, you can mandate that a user is accessing a folder from a corporate machine, for example.

Finally, attributes of the files and folders can be included in access and auditing expressions. These attributes can come from manual end user classifications, or from the File Classification Infrastructure (FCI) feature in Windows Server.

As you can see, DAC is a broad feature, and a successful deployment will require the involvement of resources across the organization, both inside and outside of IT. We'll discuss the Active Directory configuration component of this feature here, as well as a quick walk-through to validate your configuration steps. We'll leave the discussion of how to best deploy DAC and the various file server–specific features to other publications that are more qualified to discuss these topics.

Configuring Active Directory for DAC

There a couple of different paradigms for deploying DAC, which makes this discussion more complicated. You can take advantage of the compound ACL functionality with just group membership without any additional configuration. But if you want to start constructing ACLs based on attributes of the users (or their devices), you'll need to configure *claim types* in Active Directory as well as some Group Policy settings.

Finally, you can publish filesystem access policies that dynamically apply or model ACLs to filesystem data based on properties of the files themselves. These policies are known as *central access policies*. Central access policies are comprised of a series of *central access rules* that govern the ACL that will be applied to matching files and folders. The properties that can be set on files and folders are published via *resource properties*.

Fortunately, Microsoft has provided a GUI for configuring most of the DAC feature set. If you haven't already explored the Dynamic Access Control node in the Active Directory Administrative Center (ADAC), take a few minutes to do so. Each of the components we just discussed is configurable under the Dynamic Access Control node. The only component that must be configured solely via Windows PowerShell is the claims transformation functionality that is used when you need to pass claims across a forest trust.

Configuring claim types

The first task you'll likely want to take on is making attributes available as claims. This is configured by creating claim types in Active Directory. Claim types map an attribute of the user or the user's device in Active Directory to a claim that is included in the user's access token. To create a claim type, launch ADAC and browse to *Dynamic Access Control\Claim Types*. In Figure 16-20, we are creating a claim type called *Employee Type*. This claim is sourced from the `employeeType` attribute.

If there are a well-known set of values that will be in the claim, you can configure suggested values at the bottom of the screen that will be displayed in a drop-down list in the ACL editor. Figure 16-21 shows how to add a suggested value for full-time employees. The value field corresponds to the actual data that is stored in the Active Directory attribute, while the display name field is what is shown in the ACL editor. Once you create a claim type, it will be available for use on Windows Server 2012 file servers across the forest.

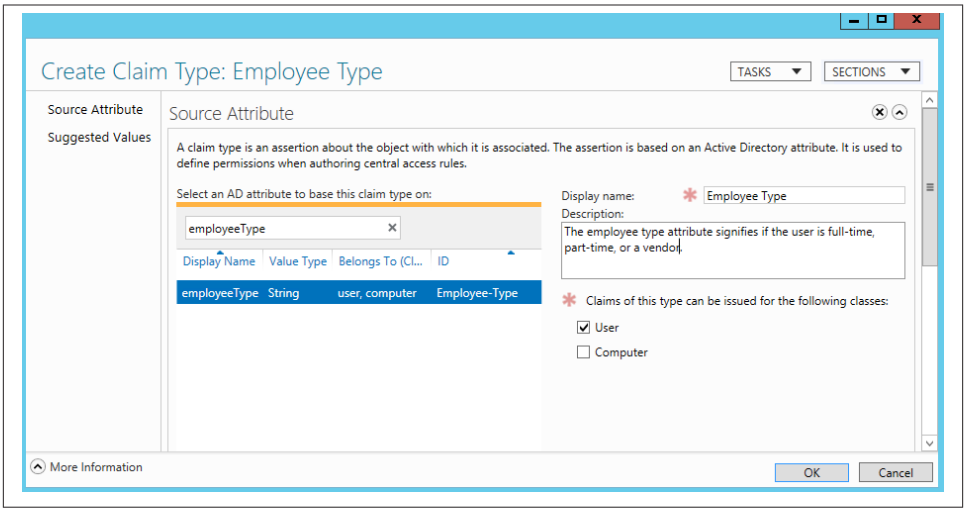


Figure 16-20. Creating the Employee Type claim

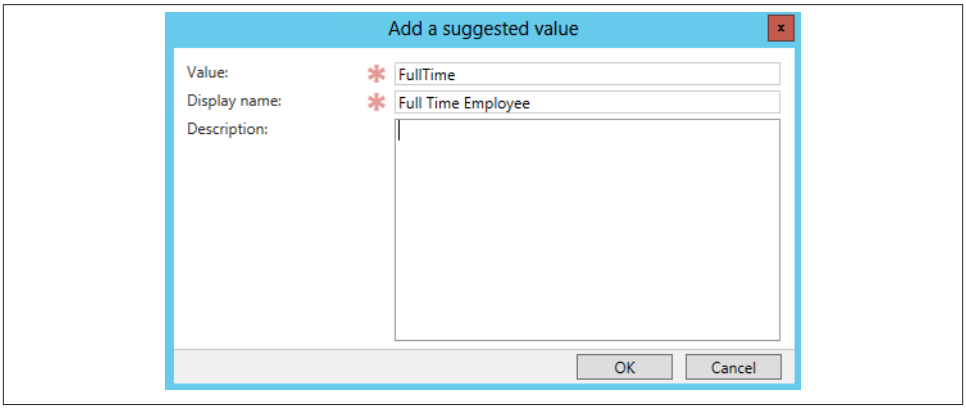


Figure 16-21. Adding a suggested value for a claim

It is very important that you think about the new dependencies that are being created on the data in the attributes that you are publishing as claims. As soon as you publish an attribute as a claim, you will not be able to change the format of the data stored in the attribute without coordinating an update to all of the ACLs that reference that attribute's format. If you are sourcing attribute values directly from another system (such as a human resources database), you will need to communicate this dependency to the owners of the source system. The values in the source system will not be able to be changed without coordination across the Active Directory and file server teams.

Configuring central access policies

Central access policies enable organizations to apply access rules to data based on properties of the data from a central location (Active Directory). This functionality is especially useful when you have a mechanism in place to classify data and specific policies or regulations to comply with. For example, you might have a policy that any data that contains personally identifiable information (PII; e.g., national/regional ID numbers) that is classified at moderate impact or higher must only be accessible to full-time employees.

To begin, use ADAC to enable the Personally Identifiable Information resource property. This is a resource property that is included out of the box with DAC. You can add as many resource properties as you like and/or customize the resource properties that are included.

Next, you need to use ADAC to define a central access rule that describes the access control you want to implement. A central access rule is comprised of scoping criteria that determine the data that the rule applies to and an ACL to apply to that data. To scope the rule to apply only to PII, edit the target resource criteria as shown in Figure 16-22.

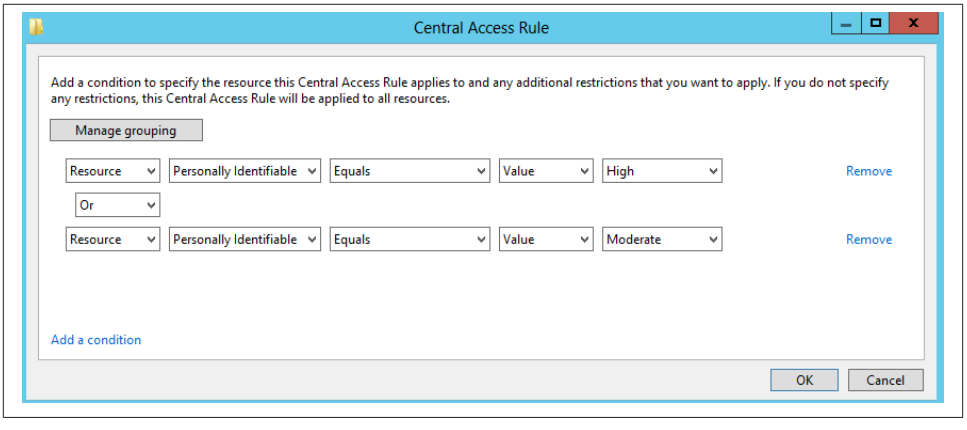


Figure 16-22. Central access rule targeting criteria

Next, configure the permissions to restrict access only to Domain Users who have the Employee Type claim with a value of Full Time Employee, as shown in Figure 16-23. You can either configure permissions to operate in a "proposed" mode, whereby resources that have permissions that don't match the policy will be logged in the event log, or enforce the permissions on all matching resources. We recommend that you begin with the audit/proposal mode to assess the impact of your policies.

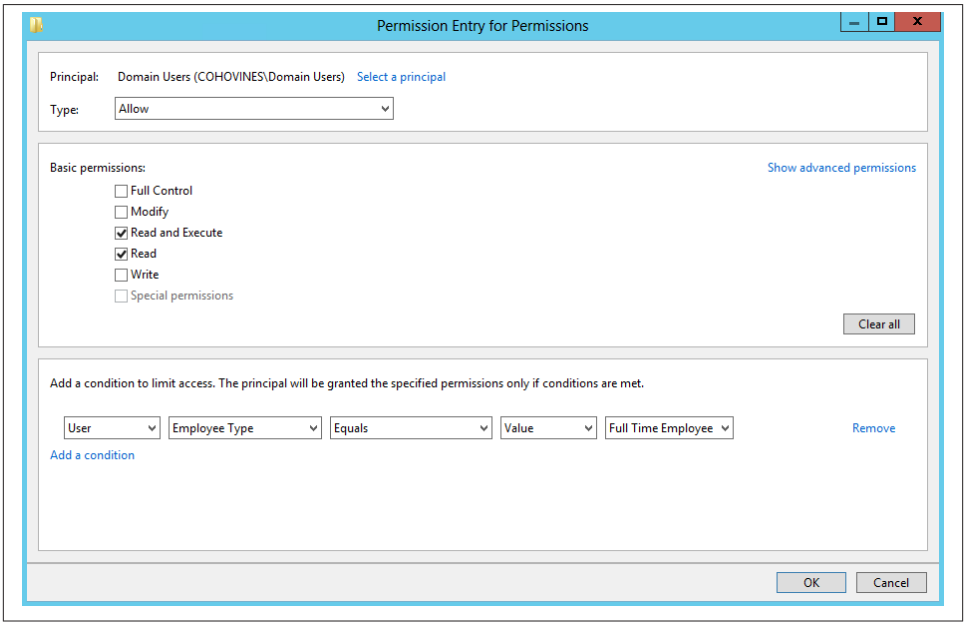


Figure 16-23. Central access rule permissions editor

When you're done, your central access rule should look similar to Figure 16-24.

Once you have created the rules you want to apply to a resource, you need to add them to a central access policy. Figure 16-25 shows a basic policy that contains the rule we just created. Once you have the policy ready to apply, you'll need to make it available to the file server via Group Policy.

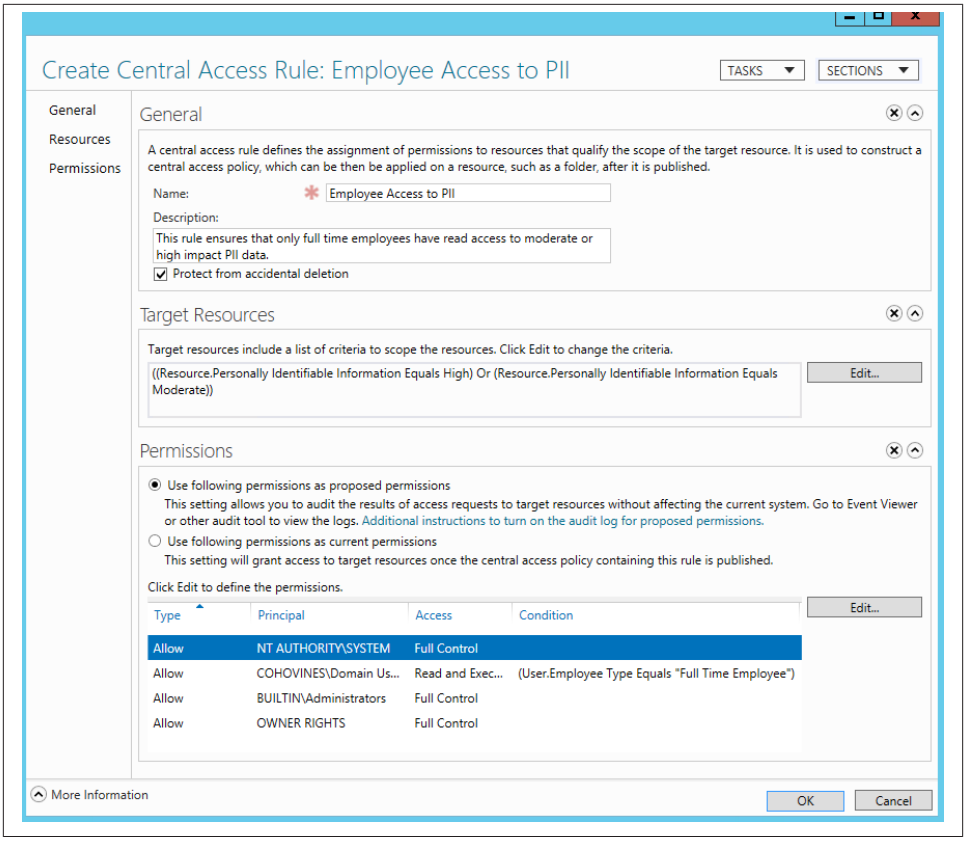


Figure 16-24. Central access rule configuration

Browse to the *Computer Configuration\Policies\Windows Settings\Security Settings\File System\Central Access Policy* node in the group policy with which you'll be applying the central access policy. You can apply one or more policies by right-clicking and selecting Manage Central Access Policies. Add the policy you created in the previous step.

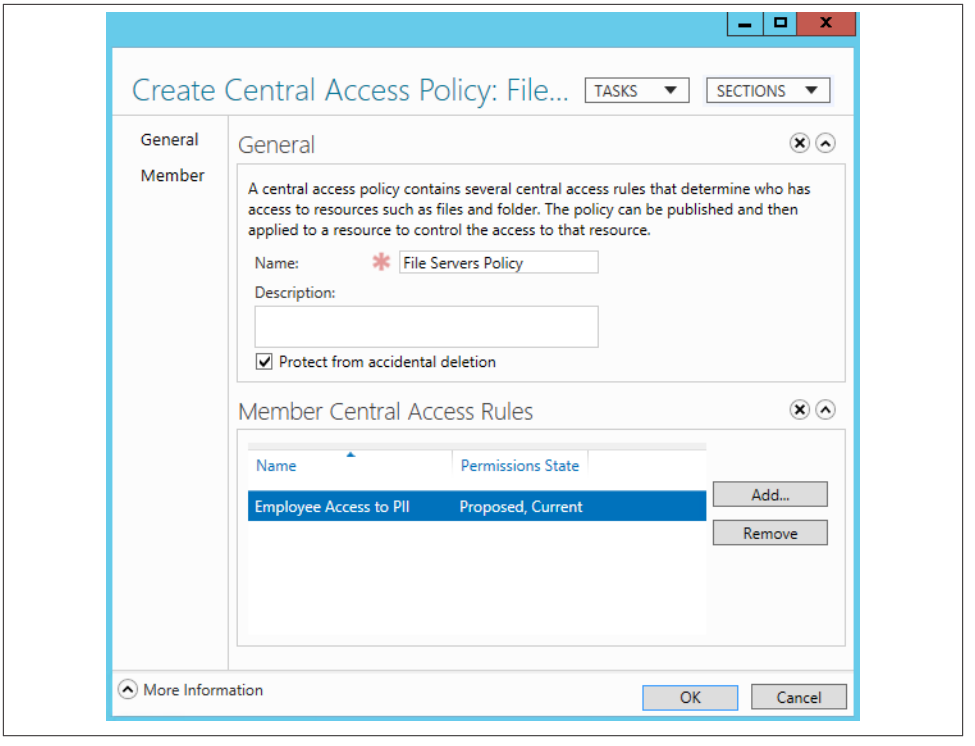


Figure 16-25. New central access policy

You'll also want to enable the central access policy auditing features to take advantage of the audit/proposal mode we selected in the central access rule. You can do this by browsing to the *Advanced Audit Policy Configuration\Audit Policies\Object Access* node under *Security Settings* in the Group Policy Management Editor. Enable success and failure auditing for the *Audit Central Access Policy Staging* and *Audit File System* entries.

Once you've published a central access policy to a file server, you must apply the policy to a folder structure. Only once you've completed this step will the rules in the policy take effect. To complete this step, open the properties of the folder you want to apply the policy to and access the Security tab. Click Advanced, switch to the Central Policy tab, and apply the policy created earlier, as shown in Figure 16-26.

Kerberos policies

If you are running in a forest that is not at the Windows Server 2012 forest functional level, you'll probably need to tweak one or more policies that apply to your domain controllers and/or file servers in order to ensure that DAC is available on your Windows Server 2012 file servers. The first policy you'll need to configure is to enable issuance of claims and compound identity on your domain controllers. Edit the Default Domain

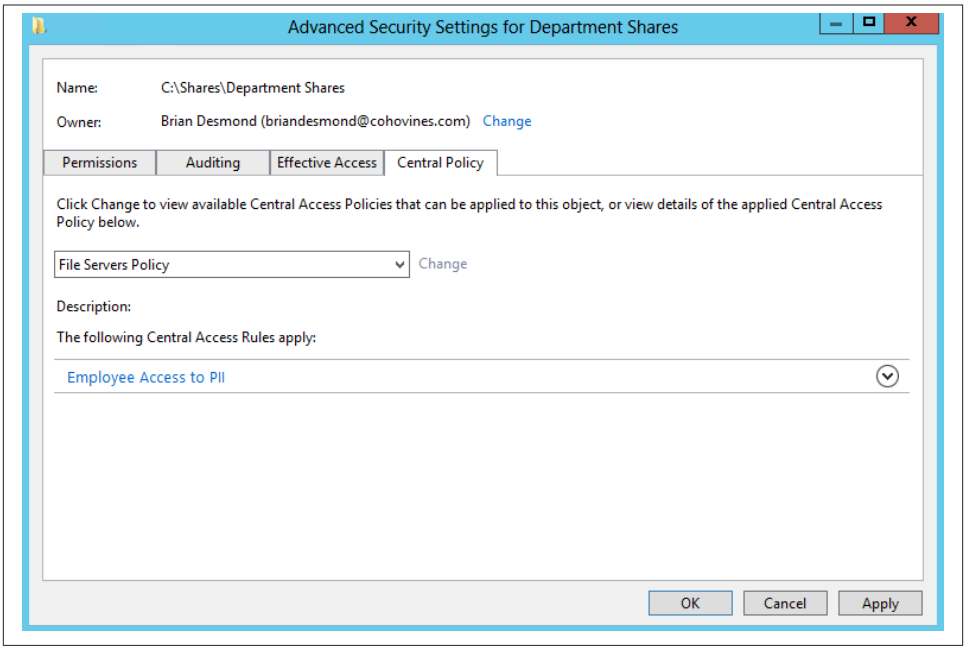


Figure 16-26. Central access policy application

Controllers Policy (or another GPO that applies to your DCs) and browse to *Computer Configuration\Policies\Administrative Templates\System\KDC*. Set the "KDC support for claims, compound authentication, and Kerberos armoring" policy to Supported.

If you are using claims in your ACLs on a file server but you have not deployed one or more central access policies to that file server, you'll need to ensure that the file server requests a service ticket on behalf of the user from a Windows Server 2012 DC when the user's token doesn't include claims information.

If you have Windows 8 clients and you want to take advantage of device claims, you'll need to enable compound identity in a GPO that applies to all of your clients. To do this, browse to *Computer Configuration\Policies\Administrative Templates\System \Kerberos* in the GPME and set the "Kerberos client support for claims, compound authentication and Kerberos armoring" policy to Enabled. You must also enable the "Support compound authentication" policy and configure the policy with the Always option.

Using DAC on the File Server

While we've looked at how to apply ACLs via a central access policy, it's also possible to simply start using claims and the new ACL functionality that DAC brings without going through the central access policy processes.

Compound expressions with groups

Out of the box, you can start creating ACLs on Windows Server 2012 file servers that simply involve existing group memberships. The power in this new feature is that you can define Boolean expressions using the *AND* and *OR* operators, as well as setting matching (such as requiring that the user is a member of one or more groups in a list).

Consider our earlier example, where we needed to restrict access to a folder to users who were members of the Finance Users group as well as the Managers group. To complete this task, open the security properties of the folder, and then click Advanced. Add the Finance Users group to the ACL and configure the permissions (e.g., read or write access). Next, add a condition that the user is also a member of the Managers group, as shown in Figure 16-27.

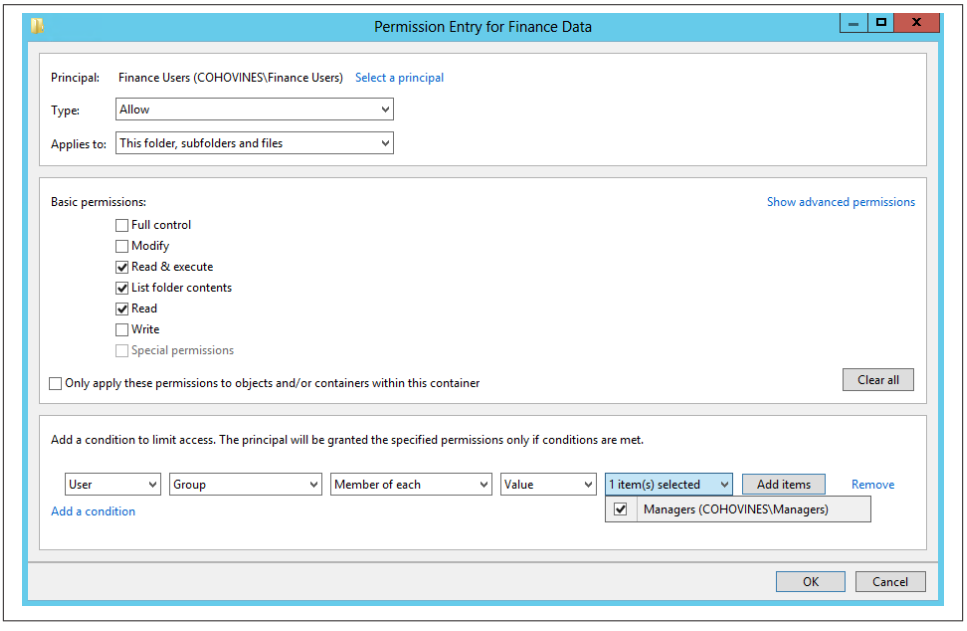


Figure 16-27. Compound group expression

While this is a simple example consisting of only two groups, you can see how you can easily construct much more complex expressions that model the structure of your organization using existing Active Directory groups. It would not be out of the question

to reduce the number of groups some users are a member of considerably just by taking advantage of this feature on your file servers.

Using claims in your ACLs

The same editor we used in Figure 16-27 can be used to construct expressions based on user and device claims as well as properties of the target resource. As a simple example, let's extend the expression we constructed earlier that allows access to members of the Finance Users and Managers groups to require that the users not be vendors.

To do this, add a rule that says Employee Type does not equal Vendor, as shown in Figure 16-28. You'll see that the drop-down list is prepopulated with the list of suggested values we defined when we created the claim type.

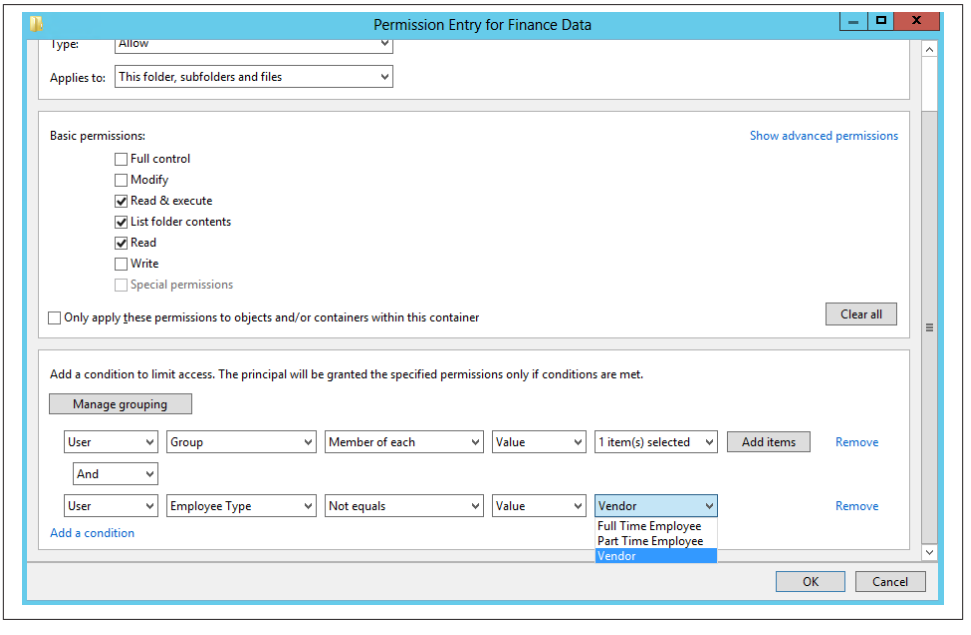


Figure 16-28. Excluding vendors from access to data

Auditing

You can use the same concepts we just explored for file access to configure auditing as well. This makes the filesystem auditing functionality in Windows substantially more useful. Previously, auditing was essentially an information overload as there was little flexibility in configuring what operations were audited.

Take into account the ability to construct expressions based on user, device, and resource properties, and you might configure an auditing rule that only audits access to resources that contain PII, as shown in Figure 16-29.

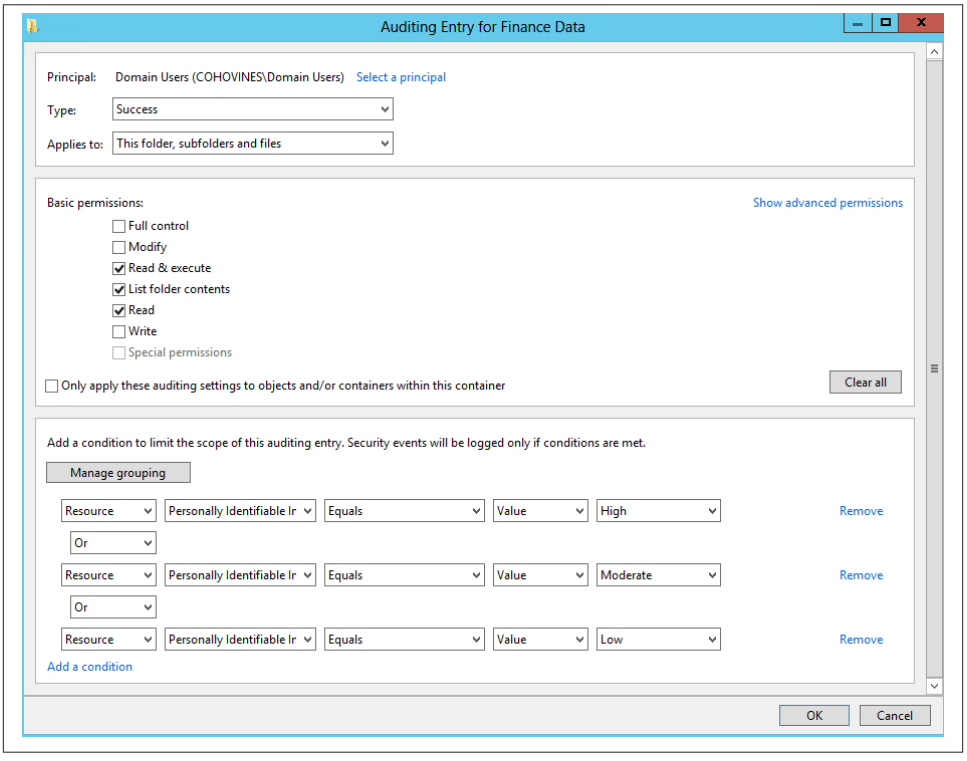


Figure 16-29. Configuring auditing of access to personally identifiable information

Summary

Security is always important, and where 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 permissions 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. Dynamic Access Control makes planning even more important, with the ability to create security dependencies on individual attribute values in Active Directory.

Although 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 showed you here should allow you to plan and implement sensible permissions schemes, which was the goal of this chapter.

CHAPTER 17

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 can potentially be stored in Active Directory.

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 will require. Because it could never anticipate every class of object or every specific attribute 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, and not being aware of this 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. Under Windows Server 2003 and newer forests, you can disable or redefine schema extensions, but you can never remove them completely.
- Although 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 was added to the Global Catalog), a full resync had to be done for all Global Catalog servers. Fortunately,

Microsoft resolved this in Windows Server 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 whether you can obtain the required design result some other way. 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 won'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 schema extension that you might 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 first look at how you would design the changes you may wish to make in an enterprise environment.

If you have not done so already, we highly recommend that you review Chapter 5 for an in-depth discussion of the Active Directory schema. This chapter is dependent on the concepts discussed there.

Nominating Responsible People in Your Organization

If you don't already have a central person responsible for the object identifier (OID) namespace for your organization, you should identify such a person. This OID manager is responsible for obtaining an OID namespace, designing a structure for the namespace that makes sense for 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 and attributes from that structure as required. Whenever a new class or attribute is to be added to your organization's schema, the OID manager will provide a unique OID for that new addition. This is then logged by the OID manager with a set of details about the reason for the request and the type of schema extension that it is to be used for. Keep in mind that aside from Active Directory, other members of your IT organization may have a requirement for OIDs. Other LDAP directories and Simple Network Management Protocol (SNMP) applications are two examples of applications that also make use of OIDs.

The Active Directory administrators, by comparison, are responsible for designing and creating proper classes in the schema for a forest as well as for implementing schema extensions required by third-party applications. They are responsible for actually

making changes to the schema via requests from within the organization, and 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 partial attribute set contains the right attributes.

If you are designing code that will modify some other organization's schema, such as for a software product, the documentation accompanying that code should make it explicitly clear exactly which classes and attributes are being created and why. The documentation also should explain that the code needs to be run with the privileges of a member of the Schema Admins group, since in many organizations the Schema Admins AD group is empty.

A better solution to programmatically processing schema changes is to supply organizations with the LDIF files for the schema modifications. This allows the organizations to review the actual changes and incorporate them into a batch update process with other schema modifications, which allows for easier testing and production implementation. Some large organizations have extensive schema change control procedures that require LDIF-format files describing all changes. Failure to supply the required LDIF file in these organizations results in the update being rejected.

Note that while the responsibilities of the OID manager do not necessarily coincide with those of Active Directory administrators, it is entirely possible that they are the same person or persons.

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 deployment will hold. After all, if you know your data, you can decide what changes you want to make and whom those changes might impact.

Designing the Data

No matter how you migrate to Active Directory, at some point you'll need to determine exactly what data you will store for the objects in Active Directory. 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 to know what languages each member of your staff speaks or what 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 attributes already exist in the Active Directory schema, and some don't. At some point, you need to determine the actual data that you want to include. This is an important decision, since not all data should necessarily be added to Active Directory. While it may be nice to have shoe size data in the directory, it may not make much business sense. You need to ask, "Is this data needed on all domain controllers for a given domain or all global catalogs in the forest?" It is possible that an application partition, Active Directory Lightweight Directory Services (AD LDS), or even a SQL Server database may be a better store for this information. Solutions outside of the AD forest are often particularly appropriate when the data in question is sensitive or confidential in nature.

Keep in mind that every piece of data that gets added to the directory needs to be replicated. This can have a significant impact on network traffic, storage needs, and the time required to build new domain controllers. If the data will change frequently, factor in the increased replication load. If replication is a burden on your network, you may experience additional latency in overall replication due to the added overhead.

To Change or Not to Change

When you identify a deficiency in your Active Directory schema, you have to look into whether modifying the schema is the correct way to proceed. 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.

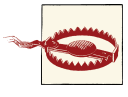
To help you make that choice, you should ask yourself whether there are any other objects or attributes that you could use to solve your problem.

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, which you are not planning 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 each of our students. In Great Britain, every university student has a University and Colleges Administration System number, more commonly known as the UCAS number; this is 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.

Since we were initially only looking to store this piece of information in addition to the default user information, we were not talking about a huge change in any case. We simply looked to see whether we could use any other schema attribute to contain the data. We soon found the `employeeID` user attribute that we were not ever intending to use, and that seemed to fit the bill, so we decided to use that. While it isn't as appropriately named as an attribute called `UCAS-Number` would be, it did mean that we didn't have to modify the base schema in this instance.

The important point here is that we chose not to modify the schema, having found a spare attribute that we were satisfied with. We could just as easily have found no appropriate attribute and decided to go through making the schema change using our own customized attribute.



This example of repurposing the `employeeID` attribute is one where the data the attribute was repurposed to hold is closely aligned with the original intent implied by the name `employeeID`. It is *not at all* advisable to repurpose attributes that are included in the base schema for purposes that are not remotely related to their original intent. It is entirely possible that a later version of Active Directory may begin using an attribute in the base schema for its original intent, or a third-party application may make this assumption. If you have elected to repurpose the attribute, you will then need to migrate the data out of this attribute and update all of the applications that depend upon it.

If you've installed Microsoft Exchange into the forest, there is also a set of attributes available for you to use for whatever you need. These are known as the `extension` or `custom attributes` and have names like `extensionAttribute1`, `extensionAttribute2`, and so on. These are not generally used by the operating system and have been left in for you to use as you wish. There are 50 created by default with Exchange Server 2013, thus giving you spare attribute capacity built right into Active Directory. So, if we wanted to store the number of languages spoken by a user, we could just store that value inside `extensionAttribute1` if we chose.

Making use of extension attributes and making use of unused attributes works well for a small number of cases. However, if there were 20, 30, or more complex attributes, each with a specific syntax, or if we needed to store 20 objects with 30 attributes each, we would have more difficulty. When you have data like that, you need to consider the bigger picture.

The Global Picture

So, you have a list of all your data, and you suspect either that the schema will not hold your data or that it will not do so to your satisfaction. You now need to consider the future of your organization's schema and design it accordingly. The following questions should help you decide how to design for each new class or attribute:

- Is this class or attribute already in the schema in some form? In other words, does the attribute already exist by default, or has someone already created it? If it doesn't exist, you can create it. If it does already exist in some form, can you make use of that existing attribute? If you can, you should consider doing so.

  If you can't, you need to consider modifying the existing attribute to cope with your needs or creating a second attribute that essentially holds similar or identical data, which is wasteful. If the existing attribute is of no use, can you create a new one, migrate the values for the existing attribute to the new one, and disable the old one? These are the sorts of questions you need to be thinking of.
- Is this a class or attribute that is to be used only for a very specific purpose, or could this object potentially be made of use (i.e., created, changed, modified) by others in the organization? If the class or attribute is for only a specific purpose, the person suggesting the change should know what is required. If the class or attribute may impact others, care should be taken to ensure it is designed to cope with the requirements of all potential users—for example, that it can later be extended if necessary without affecting the existing object instances at the moment the schema object is updated.

  For an attribute, for example, you should ask whether the attribute's syntax and maximum/minimum values (for strings or integers) are valid or whether they should be made more applicable to the needs of many. Specifically, if you created a `CASE_INSENSITIVE_STRING` of between 5 and 20 characters now and later you require that attribute to be a `CASE_SENSITIVE_STRING` of between 5 and 20 characters, you would need to create a new attribute and migrate the data as the syntax of an attribute cannot be changed once the attribute is created.
- Are you modifying an existing class by adding an attribute? If so, would this attribute be better if it were not applied directly to the object, but instead added to a set of attributes within an auxiliary class?
- Are you adding an attribute to an existing class that you normally manage through the standard GUI tools, such as Active Directory Users and Computers? The new attribute will not automatically show up in the GUI and will require changing the tool being used or extending the tool (if that is possible). You must be aware of the impact that your changes may have on existing tools and ones that you design yourself.

Basically, these questions boil down to four much simpler ones:

- Is the change that needs to be made valid and sensible for all potential uses and users of this object?
- Will the change impact any other changes that may need to be made to this and other objects in the future?
- Will the change impact anyone else now or in the future?
- Will the change impact any applications that people inside or outside the company are developing?

As when creating a valid OID namespace, make sure that your classes and attributes are created with sensible names. These names should have a unique company prefix for easy identification and consist of capitalized words separated by hyphens. For more information about choosing a prefix for your names, see the sidebar "Choosing a Sensible Naming Prefix."

Choosing a Sensible Naming Prefix

Using a unique prefix for schema extensions may not seem important at first glance. The benefit of unique prefixes comes into play if a company finds out another company is also using the same prefix. This can become extremely problematic if the other company is an application vendor.

For example, say that MyCorp Financial Services is prefixing its schema extensions with the "myCorp" prefix. MyCorp Financial has extended its schema with two new attributes: `myCorpAttrib1` and `myCorpAttrib2`. Now suppose MyCorp Financial purchases a software package from another company, MyCorp Software Solutions, who also chose to use attribute names of `myCorpAttrib1` and `myCorpAttrib2`.

In this scenario, MyCorp Financial Services would be in a very bad position. MyCorp Financial's only option would be changing all previous uses of its attributes so that the names could be reused by the application. If MyCorp Financial did not rename its attributes, it would not be able to use the application it had purchased.

An important rule of thumb is to never modify default system attributes. This ensures that you never conflict with anything considered a default by the operating system, which might eventually cause problems during upgrades or with other applications, such as Exchange. Adding extra attributes to classes is fine, but avoid modifying existing attributes.

Creating Schema Extensions

There are three ways to modify the schema: through the Active Directory Schema MMC snap-in, using LDIF files, or programmatically using ADSI or LDAP. We will not cover the Active Directory Schema snap-in very heavily here since it is fairly straightforward to use, although we will cover its use in managing the schema master FSMO role. Typically, you should not use the Active Directory Schema snap-in to extend the schema. Instead, you should use LDIF files. LDIF files are much easier to review and document, since they are simply specially formatted text files. Most vendors provide LDIF files, which contain the schema extensions that you can review and then import at your leisure.

It can be extremely tedious to create custom schema extensions by hand with an LDIF file. Instead, you may want to take advantage of the Active Directory Schema MMC snap-in in a development forest (or an AD LDS instance) to create your schema extension. Once you have created the extension with the MMC, you can export it to an LDIF file with *ldifde* or the AD Schema Analyzer and publish the LDIF file. For more information on AD LDS and the AD Schema Analyzer, refer to Chapter 20.

Running the AD Schema Management MMC Snap-in for the First Time

The Schema Management MMC snap-in 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 snap-in by typing the following command at the command prompt:

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

You can then start the Active Directory Schema console by creating a custom MMC and adding the Active Directory Schema snap-in to it. To create a console, go to Start → Run, type **`mmc.exe`**, and click OK. Then, in the empty MMC, go to File → 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 Active Directory Schema 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 master FSMO owner to allow you to modify the schema. 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 domain controller on which you wish to make the changes.

On the schema master itself, open up the registry using *regedt32.exe* or *regedit.exe* and locate the following key: `HKLM\SYSTEM\CurrentControlSet\Services\NTDS\Parameters`.

Now, create a new `REG_DWORD` value called `Schema Update Allowed` and set the value to 1. That's all you need to do. You now can edit the schema on that domain controller.

An 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., run the file) on the DC where you wish to enable schema updates:

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

This will automatically modify the registry for you, so you don't need to do it by hand. Once you've modified the registry on a particular domain controller 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 domain controller will be accepted.

The Schema Cache

Each domain controller maintains a copy of the entire schema in memory. This is known as the *schema cache*. It is used to provide a very rapid response when requesting a schema object OID from a name.

The schema cache is actually a set of hash tables of all the `classSchema` and `attributeSchema` objects known to the system, along with specific indices (`attributeID`, `mAPIId`, and `lDAPDisplayName` for `attributeSchema` objects and `governsID` and `lDAPDisplayName` for `classSchema` objects) for fast searching.

The objects are loaded into the schema cache when the domain controller is booted, and again five minutes after an update. However, if you need the schema cache to be updated immediately for some reason, say after the creation of a new object or attribute class, you can force an immediate reload of the cache.

As we said, the system holds a copy in memory solely to aid in searches that require quick and regular access to the schema. If the system were to keep the cache and the actual Active Directory schema in sync, it could be costly in terms of performance; making changes to the schema is an intensive process due to the significant checking and setting of object defaults by the system upon creation of new objects. Consequently, there is a time delay between changes made to the underlying schema and the cached copy. Typically, the schema tends to be updated in bunches. This is likely to be due to applications creating multiple classes for their own purposes during an installation or even normal operation. If classes are still being created after five minutes, the system

updates the cache in five-minute increments after the first five-minute update has completed. This continues for as long as the schema class updates continue.

During the intervening five-minute period, when the underlying schema has been modified but the cache has yet to be updated, instances of objects or attributes of the new classes cannot be created. If you try to create an object, the system will return an error. This is due to the fact that object creations refer to the cache and not the underlying schema. To get around this problem, you can force an immediate reload of the cache by updating a special operational attribute on the *RootDSE*. We'll cover this later when we consider how to use the Schema Management MMC snap-in to create and delete classes. In a similar vein, if you mark an object as defunct, this will not take effect until the cache is reloaded.

Although you cannot create new instances of new object types that you have created until the schema cache refreshes, you can add new attributes or classes that you have created to other classes that you are creating. For example, if you create a new attribute, you can immediately add it to a new class. Why? Because the attribute or class is added using an OID, and the system thus doesn't need to do any lookups in the schema cache. While all system checks by Active Directory confirming that the data is valid—covered in the section "Checks the System Makes When You Modify the Schema" (page 508)—will still be performed, the checks are performed on the schema in Active Directory, not the cache. If this weren't the case, you would have to wait for at least five minutes before any new attributes that you created could be added to new classes, and that would be unacceptable.

The Schema Master FSMO

The schema master FSMO is the domain controller where changes to the schema take place so that multiple users or applications cannot modify the schema on two or more different domain controllers at the same time. When Active Directory is installed in an enterprise, the first server in the first domain in the forest (the forest root domain) becomes the nominated schema master FSMO. Later, if changes need to be made to the schema, they can be made at the current master.

You can transfer the role from an existing schema master in three ways: via the Active Directory Schema MMC snap-in, via the *NTDSUTIL* tool, or via code that makes use of ADSI or LDAP.

Using the Active Directory Schema MMC snap-in to transfer the role is easy. First you need to connect to the server that is to be the new master, and then you need to force the role to change to the server to which you are now connected. To start the process, simply run the MMC and right-click Active Directory Schema in the lefthand scope pane. From the context menu that drops down, select Change Active Directory Domain Controller.

You can now select a new server to connect to. You should transfer any FSMO roles (not just the schema master) to a new server before shutting down a server for an extended period, such as for maintenance. Once you have connected to the intended target for the schema master FSMO role, right-click on Active Directory Domains Schema in the scope pane and select Operations Master from the context menu. A dialog box will appear, showing the current DC holding the schema master FSMO role, as well as an option to change the role to the currently connected server. Figure 17-1 shows this dialog box. Click the Change button to transfer the schema master role.

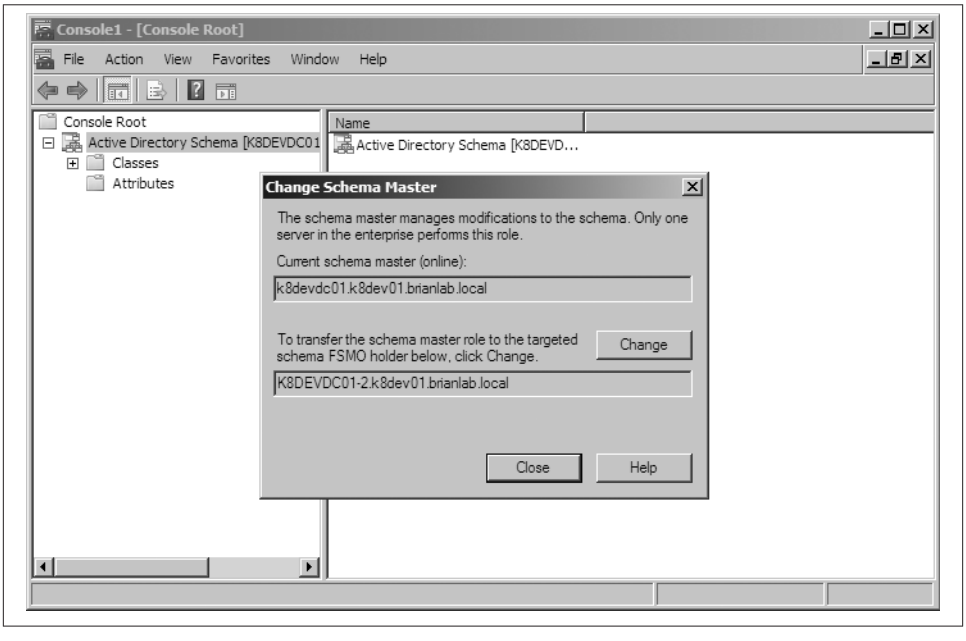


Figure 17-1. Transferring the schema master FSMO

In the event that the schema master role owner becomes permanently unavailable (due to hardware failure, operating system corruption, etc.), you will need to seize the schema master role on another domain controller. In order to seize a FSMO, you must use the *ntdsutil* command-line utility. For more information about seizing FSMO roles, see Chapter 18.

Using LDIF to Extend the Schema

One of the most commonly used ways to extend the schema is with the LDAP Data Interchange Format, or LDIF. LDIF was defined in RFC 2849 and provides a way to represent directory data via a human-readable text file. You can export data from Active Directory in LDIF format, and you can also add, modify, and delete data with LDIF. The *ldifde* program comes installed as part of any Windows Server and can be used to import and export LDIF data. To import the contents of an LDIF file, run the following command:

```
c:> ldifde -v -i -f import.ldf
```

Replace *import.ldf* with the name of the LDIF file you want to import.

LDIF files contain one or more entries, with each entry containing one or more attributes that should be added, replaced, or removed. The format is straightforward, but very strict. The following is an example of an LDIF file that would add a group object to the Users container:

```
dn: cn=mygroup,cn=users,dc=mycorp,dc=com
changetype: add
objectclass: group
description: My Group
member: cn=administrator,cn=users,dc=mycorp,dc=com
member: cn=guest,cn=users,dc=mycorp,dc=com
```

The first line must be the DN of the object. The second line is the `changetype`, which is one of `add`, `modify`, `modrdn`, or `delete`. When using `add`, as in this case, we must specify all the mandatory attributes for the object. For group objects, we need to specify only `objectClass`. The `cn` attribute is not required because it is already specified as part of the DN.

Windows 2000 Active Directory also requires the `sAMAccountName` attribute to be specified for new users and groups.

It is easy to create portable schema extensions using LDIF files. Simply create an LDIF file with all the necessary `classSchema` or `attributeSchema` object additions or modifications, and administrators using any LDIF-based client (such as *ldifde*, which ships with Windows) can easily import it into Active Directory. The following LDIF file shows how to create an attribute and an auxiliary class that contains the new attribute:

```
dn: cn=myCorp-ITUserBuilding,cn=schema,cn=configuration,dc=X
changetype: add
attributeID: 1.2.3.4.111.1
attributeSyntax: 2.5.5.1
oMSyntax: 127
isSingleValued: TRUE
lDAPDisplayName: myCorp-ITUserBuilding
objectClass: attributeSchema

dn:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-

dn: cn=myCorp-ITUser,cn=schema,cn=configuration,dc=X
changetype: add
objectclass: classSchema
description: Class for MyCorp Employees
lDAPDisplayName: myCorp-ITUser
governsID: 1.2.3.4.111.2
objectClassCategory: 3
subClassOf: top
mayContain: myCorp-ITUserBuilding

dn:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-
```

To import this sample, you need to replace the placeholder DNs with the DN of the Schema NC. You can use the *-c* switch and *ldifde*'s macro expansion capability to easily do this:

```
>ldifde -i -f schema-extension.ldf -c "cn=schema,cn=configuration,dc=x"
"#schemaNamingContext"
```

As we mentioned, all mandatory attributes for `attributeSchema` and `classSchema` objects must be specified. The order of the additions is also important. Because we wanted to add the new attribute to the class, we needed to create it first. We also needed to reload the schema cache before attempting to reference the new attribute by `lDAPDisplayName`, or a failure would have occurred. To accomplish that, we perform a modify operation against the `RootDSE` (i.e., blank DN) and write to the `schemaUpdateNow` attribute.

We could have skipped the schema cache refresh after the attribute creation if we had used the `attributeID` OID 1.2.3.4.111.1 in place of the `lDAPDisplayName myCorp-ITUserBuilding` in the `mayContain` attribute of the `myCorp-ITUser` class. Since we used the `lDAPDisplayName` in this example, it required the schema cache to be reloaded because a lookup to retrieve the proper OID had to be performed.

The benefits of using LDIF to implement schema extensions are twofold. First, since LDIF is human-readable with a well-defined syntax, it is easy for those who need to implement the extensions to see what is going to be done. If you use a program whose source the administrator cannot view, he will not have as much visibility into what changes are made. Along the same lines, LDIF files provide a crude documentation mechanism for schema extensions. Because LDIF files are just text-based files, schema administrators can archive the files on a server and have instant access to exactly what changes were made for certain applications.

Checks the System Makes When You Modify the Schema

When you create a new class or attribute, the system performs some basic checks within Active Directory to see if the data is valid, in addition to any checks you provide. The checks for attributes are shown in Table 17-1, and those for new classes are shown in Table 17-2.

Table 17-1. System checks made when creating new attributes

Attribute	System check performed
`lDAPDisplayName`	Must be unique in Active Directory.
`attributeId`	Must be unique in Active Directory.
`mapiId`	If present, must be unique in Active Directory.
`schemaIDGUID`	Must be unique in Active Directory.
`attributeSyntax`	Must correlate with `oMSyntax`.
`oMSyntax`	Must correlate with `attributeSyntax`.
`rangeLower`	If `rangeUpper` is present as well, the following should be true: `rangeUpper > rangeLower`.
`rangeUpper`	If `rangelower` is present as well, the following should be true: `rangeUpper > rangeLower`.
`linkID`	Must be unique in Active Directory. Back links must have corresponding forward links.
`searchFlags`	Ambiguous name resolution (ANR) attributes must be Unicode or Teletex.

Table 17-2. System checks made when creating new classes

Attribute	System check performed
`lDAPDisplayName`	Must be unique in Active Directory.
`governsId`	Must be unique in Active Directory.
`schemaIDGUID`	Must be unique in Active Directory.
`subClassOf`	Checks to make sure that the X.500 specifications are not contravened (i.e., that an auxiliary class cannot inherit from a structural class, and an abstract class can only inherit from another abstract class). All classes defined in this attribute must already exist.
`rDNAttID`	Must have a Unicode string as its syntax.
`mayContain`	Before you can specify a class in the `mayContain` attribute, it must already be defined in the schema.
`systemMayContain`	All classes defined in this attribute must already exist.
`mustContain`	All classes defined in this attribute must already exist.
`systemMustContain`	All classes defined in this attribute must already exist.
`auxiliaryClass`	All classes defined in this attribute must already exist and must have an `objectClassCategory` indicating either `88-Class` or `Auxiliary`.
`systemAuxiliaryClass`	All classes defined in this attribute must already exist and must have an `objectClassCategory` indicating either `88-Class` or `Auxiliary`.
`possSuperiors`	All classes defined in this attribute must already exist and must have an `objectClassCategory` indicating either `88-Class` or `Structural Class`.
`systemPossSuperiors`	All classes defined in this attribute must already exist and must have an `objectClassCategory` indicating either `88-Class` or `Structural Class`.

Making Classes and Attributes Defunct

Active Directory does not allow you to delete objects from the schema. However, if your forest is running at the Windows Server 2003 or better functional level, you can redefine or disable classes and attributes. This allows you to correct potential mistakes you may have made or to repurpose classes or attributes you are no longer using.

If you create a class or attribute of some sort and decide that you don't want it any more, you can simply disable it, which is otherwise known as making it *defunct*. This is achieved by setting the `isDefunct` attribute on the schema object to TRUE or by unchecking the "Attribute is active" or "Class is active" checkbox in the Active Directory Schema MMC snap-in.

For this to succeed for an attribute, the system makes sure that the attribute is not a mandatory or optional attribute of any non-defunct class and is not the RDN attribute (`rDNAttID`) for any class. For this to succeed for a class, the system makes sure that the class is not a parent of any other non-defunct class, is not an auxiliary class to any other non-defunct class, and is not a possible superior of any other non-defunct class. If you later decide that you want to use the schema object again, set the value of `isDefunct` to

FALSE. The checks that occur when doing this are the same as for creating a new schema object of the appropriate type in the first place.

In order to view defunct attributes and classes in the Active Directory Schema MMC snap-in, you must enable the display of these attributes and classes. To display defunct schema objects, click View→Defunct Objects.

When an attribute or class is marked as defunct, the corresponding data stored in Active Directory is not removed. With this in mind, prior to marking a class as defunct, you should remove all instances of that class from the directory. Likewise, prior to marking an attribute as defunct, you should clear the attribute on all objects.

When a schema object is defunct, attempts to create instances of it fail as if it doesn't exist. The same applies to modifying existing instances, whether of an attribute on an object or an object itself, as they will appear not to exist. You can, however, delete objects that are instances of defunct classes. Searches for objects that are instances of defunct classes will happily succeed, as will searches on non-defunct classes that contain defunct attributes. All attributes, defunct or not, can be read. This is all required to enable the administrator or application author to clean up and remove the now-defunct object instances and all values from now-defunct attributes.

Even though a schema object is defunct, it still exists in terms of its `distinguishedName`, OID, and `lDAPDisplayName`. You cannot create a second schema object that has these values, but in most cases, you can change them. The exception to this is that for an attribute used as the RDN attribute for an `objectClass`, you cannot reuse the OID.

Mitigating a Schema Conflict

Some applications that extend the Active Directory schema have done so in a way that doesn't conform with the best practices discussed in this chapter. In the most egregious cases, third parties used arbitrary OIDs from Microsoft's OID space. When administrators imported the schema extensions, they succeeded and everything appeared to be just fine. Unfortunately, Microsoft later used the same OIDs (as they were Microsoft's to begin with), and steps such as Active Directory schema upgrades during *adprep* when upgrading the forest failed.

Prior to Windows Server 2008, the LDIF files used by *adprep* could be edited to mitigate a failure due to a conflict. While this alleviated the immediate problem, it led to a forest with a base schema that was divergent from the schema Microsoft ships with each version of Active Directory. To combat this problem, Microsoft changed the *adprep* process

in Windows Server 2008 to require that the LDIF files be digitally signed. Digital signatures ensure that files have not been altered or tampered with.

The signatures for the Active Directory schema LDIF files are verified with an accompanying catalog file. *ldifde* includes a *-$* parameter that specifies the catalog file to verify the LDIF file signatures against.

Since the LDIF files for the base Active Directory schema are now signed, it's not possible to simply modify them in advance of running *adprep* to ensure the process will succeed. Instead, you will need to first mark the conflicting classes or attributes as defunct, recreate them with valid OIDs, and then repopulate the attributes or recreate the objects. Once this process is complete, you will be able to successfully complete the *adprep* process.

Summary

The importance of carefully designing the changes that you make to the Active Directory schema cannot be stressed highly enough for large corporations or application developers. Extending the Active Directory schema is not something that you should fear as an administrator or as an organization, but it is something that you should think through and plan properly. Following the guidance outlined in Chapter 5 as well as this chapter will ensure that your schema extensions are well planned. In the event that something does turn out to not be well planned, you can use the schema's defunct functionality to disable and hide classes or attributes so that you can reuse their names and unique identifiers.

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 Active Directory as a whole. While you should be cautious when modifying Active Directory, sensible administrators should have as little to fear from the Active Directory schema as they do from the Windows Registry.

CHAPTER 18

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. You're much more likely to need to restore accidentally deleted objects than to have to deal with complete corruption, and thus you should be prepared for this as well. 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? These are all stressful scenarios: clients are complaining or an application is no longer working correctly, and people aren't happy. It is during times like these that you don't want to have to scramble to find a solution. Having well-documented and tested 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 Active Directory from a backup, as well as how to recover deleted objects from the Active Directory Recycle Bin. 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.

Backing Up Active Directory

Backing up Active Directory is a straightforward operation. It can be done using the NT Backup utility provided with Windows 2000 and 2003, the Windows Server Backup utility provided with Windows Server 2008 and newer, or a third-party backup package. Fortunately, you can back up Active Directory while it is online, so you do not have to worry about creating outages just to perform backups like you do with some other systems.

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 Windows 2000 or Windows Server 2003 domain controller, the system state contains the following:

Active Directory
: This includes the files in the NTDS folder, which 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 is the *SYSVOL* share that contains all of the file-based GPO information as well as the *NETLOGON* share, which typically contains user logon and logoff scripts.

Certificate Services
: This applies only to DCs that are running Certificate Services.

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

Windows Server 2008 and newer domain controller system state backups include practically every operating system file in addition to the components listed previously. Consequently, you will notice a substantial increase in the size of system state backups on Windows Server 2008 and newer DCs.

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. If the Active Directory Recycle Bin is enabled, a deleted object will continue to reside in Active Directory for the duration of the deleted object lifetime followed by the tombstone lifetime. The deleted object lifetime dictates how long Active Directory keeps objects in the Recycle Bin for recovery. The tombstone lifetime value dictates how long Active Directory keeps deleted objects after they have

exceeded the deleted object lifetime before completely removing them. Both lifetimes are configurable. The deleted object lifetime is defined in the attribute `msDS-DeletedObjectLifetime`, while the tombstone lifetime is defined in the `tombStoneLifetime` attribute on the following object:

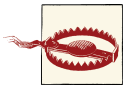
```
cn=Directory Services, cn=Windows NT, cn=Services, cn=Configuration, <ForestDN>
```

The default values for `tombStoneLifetime` for new forests in the different OS versions are documented in Table 18-1. By default the `msDS-DeletedObjectLifetime` attribute is not populated, so by default it will use the same value as your `tombstoneLifetime`. Both of these values can be modified, so you should check to see what the values are for your forest. As far as backups go, you should not restore a backup that is older than the lesser of the deleted object lifetime and the tombstone lifetime, because deleted objects will be reintroduced. If, for whatever reason, you are not able to get successful backups within the lesser of these two values, consider increasing them.

Table 18-1. Tombstone lifetime for new forests

Operating system	Tombstone lifetime
Windows 2000	60 days
Windows Server 2003	180 days
Windows Server 2003 R2	60 days (this was a regression bug in R2)
Windows Server 2003 SP2	180 days
Windows Server 2008	180 days
Windows Server 2008 R2	180 days
Windows Server 2012	180 days

Another issue to be mindful of in regard to how long you keep copies of your backups has to do with passwords. Computer accounts (and managed service accounts) change their passwords every 30 days by default. Domain Controllers will accept the current password as well as one previous password, so if you restore computer objects from a backup that is older than 60 days, those computers will more than likely have to have their passwords reset. Trust relationships can also be affected. As with computer accounts, the current and previous passwords are stored with the trust objects, but unlike computer account passwords, trust passwords are changed every seven days. That means if you authoritatively restore trust objects from a backup that is older than 14 days, you may need to reset the trusts.



Disk imaging is specifically not supported for domain controller backups. This covers all instances of imaging, from backup of virtual hardware disks to using disk image software like Ghost to special imaging available in various attached storage products. Active Directory is a distributed system running across multiple domain controllers, where each domain controller maintains state for other domain controllers. Imaging various pieces of the distributed system and recovering them separately can have catastrophic results on the consistency of the directory as a whole. Some of the possible issues are USN rollback, lingering objects, and SID rollback.

The only safe way to use disk imaging is to shut down every domain controller in the forest and then image the disks. When you need to restore, you again shut down every domain controller and roll each domain controller back to the same point in time. Even though this is a safe way to do it, it still isn't supported by Microsoft and should only be used in a test environment.

See MS Knowledge Base article 875495 for more details.

Using the NT Backup Utility

The NT Backup utility is installed on all Windows 2000 and Windows Server 2003 machines. It is available by going to Start→All Programs→Accessories→System Tools→Backup. You can also start it up by going to Start→Run, entering **`ntbackup`**, and clicking OK. Figure 18-1 shows the first screen of the NT Backup utility under Windows Server 2003.

The NT Backup utility can be used to back up the system and also to perform a restore. We will cover restores later in this chapter. If you click on the Advanced Mode link on the first screen, you'll then see a screen such as that in Figure 18-2.

In this case, we clicked on the Backup tab and then selected the box beside System State. We could also back up any of the other drives if we wanted, but the system state is all that is required when doing a basic restore of Active Directory.

By clicking the Start Backup button, we can kick off the backup. In Figure 18-2, we have configured the *D:* drive to be the location where the backup file gets stored. This could have been a remote file server or other backup media if we wanted.

We can also schedule a backup to run at an interval of our choosing by clicking the Start Backup button and then the Schedule button. After that, we click the Properties button and the screen shown in Figure 18-3 pops up.

Figure 18-1. NT Backup or Restore Wizard

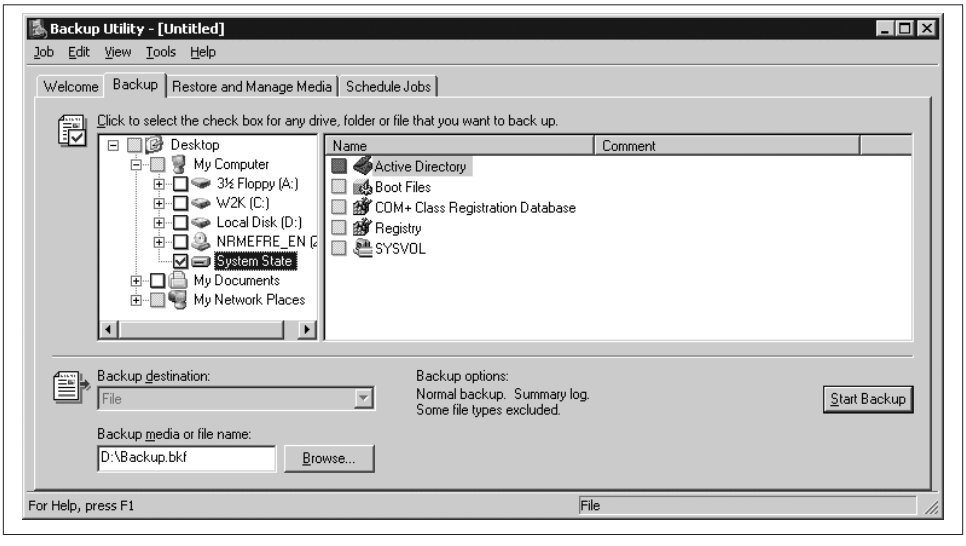


Figure 18-2. Advanced mode NT backup

In this case, we've configured the backup to run once a day at 7:30 A.M. The screen in Figure 18-3 is actually part of Scheduled Tasks, which is the job scheduling system available in Windows 2000 and Windows Server 2003.

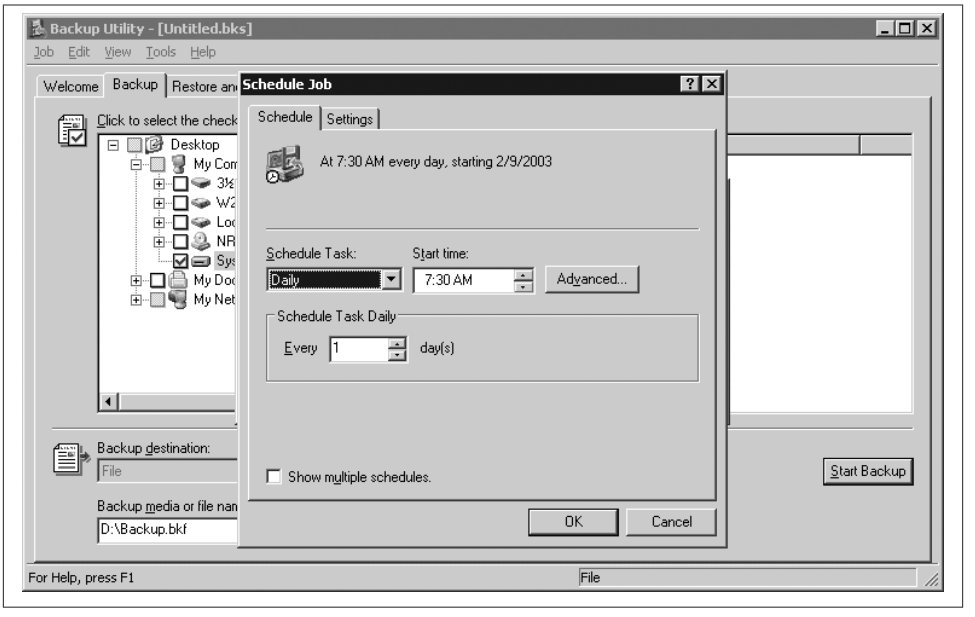


Figure 18-3. Scheduling NT Backup

Using Windows Server Backup

Windows Server 2008 introduced a completely revamped backup and restore utility. If you're used to NT Backup, you'll have some adjusting to do as the tool has been completely redesigned and has some significant new limitations. In the interest of space, we'll only discuss the functionality specific to backing up and restoring Active Directory in this book. For a full introduction to Windows Server Backup (WSB), visit this website.

Windows Server Backup is not installed by default. In order to use it, you must install the feature using Server Manager.

To launch WSB, go to Start → Administrative Tools → Windows Server Backup. The tool has been converted to an MMC snap-in as shown in Figure 18-4.

You can launch WSB directly by going to Start→Run→**`wbadmin.msc`** instead of browsing for it on the Start menu.

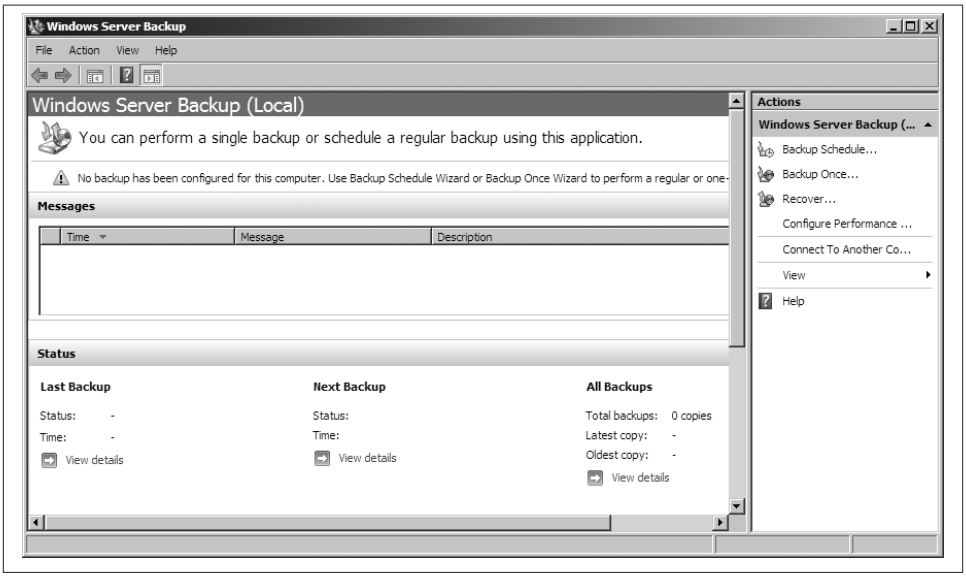


Figure 18-4. Windows Server Backup

Windows Server Backup must back up to a volume that does not contain any components of the system state backup. This means that you cannot select a volume that contains the system files, the Active Directory database (*ntds.dit*), Sysvol, Active Directory transaction logs, and so forth. The backup will be created in a directory structure on the drive specified under *<Drive>:\WindowsImageBackup\<HostName>\Backup <time stamp>*. For information about working around this limitation, see the upcoming sidebar "Allowing System State Backups to Any Volume" on page 506. If you do not have this setting enabled and you specify a volume that is not in scope for the backup, you will receive an error similar to Figure 18-5. You can alternatively specify a network path to back up to.

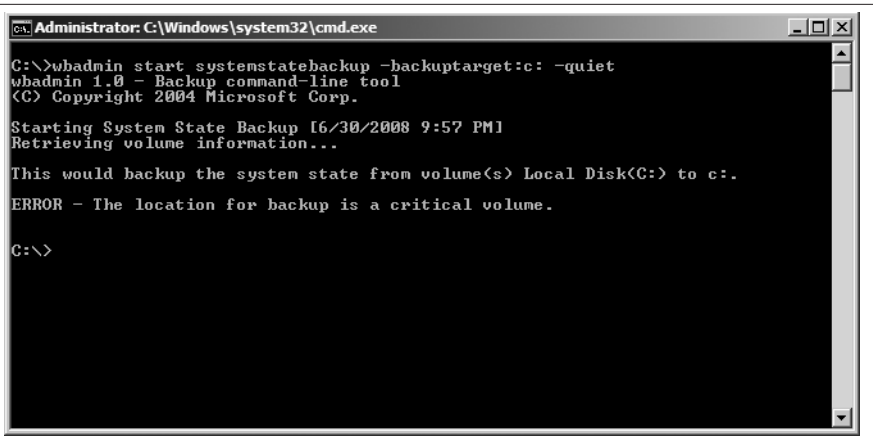


Figure 18-5. Windows Server Backup error

Allowing System State Backups to Any Volume

By default you cannot create a system state backup on a volume that includes files that are part of the backup. There is, however, a registry value you can set to work around this. You will also need to use twice the amount of space, as Volume Shadow Copy (see "Working with Snapshots" on page 525) will create a shadow copy of the backup.

To enable this functionality, create a `REG_DWORD` called `AllowSSBToAnyVolume` under `HKLM\SYSTEM\CurrentControlSet\Services\wbengine\SystemStateBackup`. Set the value of `AllowSSBToAnyVolume` to `1` to enable system state backups to a volume that is included in the backup.

For more information on this registry setting, see *http://support.microsoft.com/kb/944530*.

Due to the risks of making this change, we recommend that if you are going to use Windows Server Backup on your domain controllers you set up your partition scheme specifically to have a separate volume to store backups on.

To create a system state backup, select the Custom backup configuration in the backup wizard. Next, click Add Items and select "System state." You may also wish to select files, such as non-AD-integrated DNS zones or any other data that is specific to the domain controller. On the next page, select a destination, such as a network share or another local drive, and then complete the wizard. On our test systems, with a freshly created Active Directory forest on Windows Server 2008 R2 Enterprise, the total system state backup size was about 10 GB.

What if you want to create a system state backup from the command line? The command to create a backup stored on the *E:* drive is *wbadmin start systemstatebackup -backuptarget:E: -quiet*. If you don't specify *-quiet*, you will be prompted to continue before the backup completes.

For more information on Windows Server Backup and Active Directory, see the "AD DS Backup and Recovery Step-by-Step Guide" at this link.

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 and generally should be insignificant. But 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 fail over 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 a backup.

Does the domain controller own 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 may need to seize that role on another domain controller. Check out the "FSMO Recovery" on page 533 for more information.

Is the domain controller a Global Catalog server?
: The Global Catalog function can be added to any domain controller. But if you have only one Global Catalog server in a site and it becomes unavailable, it can impact users' ability to log in. As long as clients can access a Global Catalog, even if it isn't in the most optimal location, they will be able to log in. If a site without a Global Catalog for some reason loses connectivity with the rest of the network, this will impact users' ability to log in. With Windows Server 2003 and newer, you can enable universal group caching on a per-site basis to limit this potential issue, but only if the user is not using a `userPrincipalName` for authentication.

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, relying on that specific domain controller?
: Exchange is a heavy consumer of Active Directory, especially the Global Catalog. Failure of a domain controller that Exchange is using can cause considerable issues in the mail environment, depending on the versions of Outlook and Exchange being used. More recent versions of Exchange and Outlook (2003 and newer) handle outages better than older versions. During the outage period, mail delivery could be impacted along with client lookups. Exchange is just one example, but it illustrates an issue you have to be aware of 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 some 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.

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 either did not have a recent backup of the machine or didn't want to go through the process of restoring the DC from a backup. 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. Remove the failed DC from AD. The remnants of the old domain controller must be removed from Active Directory before you promote the freshly installed server. We describe the exact steps to do this shortly.
2. Rebuild the OS. Reinstall the operating system and any other applications you support on your domain controllers.
3. Promote the server. After you've allowed time for the DC removal process to replicate throughout the forest, you can then promote the new server into AD.
4. Configure any necessary roles. If the failed server had any FSMO roles or was a global catalog or a DNS server, you can configure the new server to have these roles.

One possible best practice is to keep a spare server that already has the OS and any other software installed ready to ship or onsite at all locations. That way, if you have a major failure with one of your domain controllers, you can use the spare server without needing to stress over getting the hardware replaced immediately in the failed machine. Alternatively, just have additional domain controller capacity in the primary sites where failures would be most painful, especially for Exchange. This alternative strategy is generally much more common than keeping spare hardware in every location.

The biggest potential drawback with this method is the restore time. Depending on the size of your DIT file and how fast your network connections are between the new DC and the server it will replicate with, the restore time could be several hours or even days. Restore time can be dramatically reduced with the Install from Media option, which allows you to take files from a system state backup from one domain controller and use them to quickly promote another domain controller. It may be faster to compress these backup files and then copy them to the remote site over the network, or ship them to the site on some other media, rather than trying to replicate the entire DIT over the WAN. For more information on this approach see the section "Install from Media" on page 512, later in this chapter.

Manually removing a domain controller from Active Directory

One of the key steps in the restore-from-replication method is removing the objects that are associated with the domain controller before it gets added to AD again. Windows Server 2008 allows you to remove a failed domain controller using the Active Directory Users and Computers MMC snap-in. Select the computer object representing the failed domain controller and delete it. You will receive a prompt similar to Figure 18-6 asking you to confirm that the domain controller is, in fact, permanently offline. Once you confirm this, the metadata cleanup steps will be performed automatically.

Under Windows Server 2003, this is a three-step process. The first step is to remove the associated metadata with the *ntdsutil* utility. The following example shows the commands necessary to remove the *DC3* domain controller, which is in the RTP site, from the *emea.mycorp.com* domain:

```
C:\> ntdsutil
ntdsutil: metadata cleanup
metadata cleanup: connections
```

Next, we need to connect to an existing domain controller in the domain that contains the DC we want to remove. In this case, we connect to *DC2*:

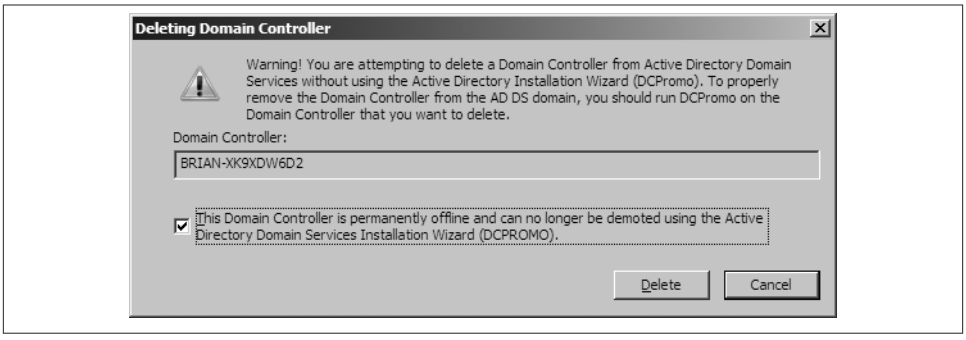


Figure 18-6. Windows Server 2008 removing a failed domain controller

```
server connections: connect to server dc2
Binding to dc2 ...
Connected to dc2 using credentials of locally logged on user.
server connections: quit
metadata cleanup: select operation target
```

Now we need to select the domain the domain controller is in. In this case, it is *emea.mycorp.com*:

```
select operation target: list domains
Found 2 domain(s)
0 - DC=mycorp,DC=com
1 - DC=emea,DC=mycorp,DC=com
select operation target: select domain 1
No current site
Domain - DC=emea,DC=mycorp,DC=com
No current server
No current Naming Context
```

Next we must select the site the domain controller is in. In this case, it is the RTP site:

```
select operation target: list sites
Found 4 site(s)
0 - CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=mycorp,DC=com
1 - CN=RTP,CN=Sites,CN=Configuration,DC=mycorp,DC=com
2 - CN=SJC,CN=Sites,CN=Configuration,DC=mycorp,DC=com
3 - CN=NYC,CN=Sites,CN=Configuration,DC=mycorp,DC=com
select operation target: select site 1
Site - CN=RTP,CN=Sites,CN=Configuration,DC=mycorp,DC=com
Domain - DC=emea,DC=mycorp,DC=com
No current server
No current Naming Context
```

After listing the servers in the site, we must select the server we want to remove. In this case, it is *DC3*:

```
select operation target: list servers in site
Found 3 server(s)
0 - CN=DC1,CN=Servers,CN=RTP,CN=Sites,CN=Configuration,DC=mycorp,DC=com
1 - CN=DC2,CN=Servers,CN=RTP,CN=Sites,CN=Configuration,DC=mycorp,DC=com
2 - CN=DC3,CN=Servers,CN=RTP,CN=Sites,CN=Configuration,DC=mycorp,DC=com
select operation target: select server 2
Site - CN=RTP,CN=Sites,CN=Configuration, DC=mycorp,DC=com
Domain - DC=emea,DC=mycorp,DC=com
Server - CN=DC3,CN=Servers,CN=RTP,CN=Sites,CN=Configuration,DC=mycorp,DC=com
   DSA object - CN=NTDS Settings,CN=DC3,CN=Servers,CN=RTP,CN=Sites,
                CN=Configuration,DC=mycorp,DC=com
   Computer object - CN=DC3,OU=Domain Controllers,DC=emea,DC=mycorp,DC=com
No current Naming Context
select operation target: quit
```

This process has been considerably simplified in Windows Server 2003 Service Pack 1; however, you need to know the `distinguishedName` of the Domain Controller's `server` object in the Configuration container. It is recommended that you simply follow the preceding directions for removing dead domain controllers, as there is less possibility of a mistake.

The last step removes the metadata for the selected domain controller:

```
metadata cleanup: remove selected server
```

At this point, you should receive confirmation that the DC was removed successfully. If you receive an error stating that the object could not be found, it might have already been removed if you tried to demote the server with *dcpromo*.

If you are performing this procedure on a domain controller that is running a version of Windows prior to Windows Server 2003 SP1, you will then need to manually remove a few more objects from Active Directory. See MS Knowledge Base article 216498 for details on this. You will need to manually remove the server object under the site in the configuration partition regardless of which version of Windows you are running .

Restore from Backup

Another option to reestablish a failed domain controller is to restore the machine using a backup. This approach, known as a *nonauthoritative restore*, does not require you to remove any objects from Active Directory. When you restore a DC from a backup, the latest changes will replicate to make it current. If time is of the essence and the backup file is immediately available, this will be the quicker approach because only the latest changes (those made since the last backup), instead of the whole directory tree, will be replicated over the network.

Here are the steps to restore from a backup:

1. Rebuild the OS. Reinstall the operating system and any other applications you support on your domain controllers. Leave the server as a standalone or member server.
2. Restore from the backup. Use your backup package to restore at least the system state onto the machine. In the next section "Restoring Active Directory" on page 516, we will walk through the NT Backup and Windows Server Backup utilities to show how this is done.
3. Reboot the server and allow replication to complete. If the failed server had any FSMO roles or was a GC, you can configure the new server to have these roles.

It is also possible to restore the backup of a machine onto a machine that has different hardware. Here are some issues to be aware of when doing so:

- The number of drives and drive letters should be the same.
- The disk drive controller and configuration should be the same.
- The attached cards, such as network cards, video adapters, and processors, should be the same. After the restore, you can install the new cards, which should be recognized by Plug and Play.
- The *boot.ini* from the failed machine will be restored, but it may not be compatible with the new hardware, so you'll need to make any necessary changes.
- If the hardware abstraction layer (HAL) is different between machines, you can run into problems. For example, if the failed machine was single-processor and the new machine is multiprocessor, you will have a compatibility problem. The only workaround is to copy the *Hal.dll*, which is not included as part of system state, from the old machine and put it on the new machine. The obvious drawback to this is it that will make the new multiprocessor machine act like a single-processor machine.

Because there are numerous things that can go wrong with restoring to different hardware, we highly suggest you test and document the process thoroughly; refer to MS Knowledge Base article 263532. The last thing you want to have to do is troubleshoot hardware compatibility issues when you are trying to restore a crucial domain controller.

Install from Media

One of the challenges of working with Active Directory in a highly distributed environment is often the impact of replicating the initial copy of the database when a new domain controller is promoted. Over a slow WAN link with a large database, this operation can easily take days or weeks. While this is running it can also put an unmanageable load on WAN links that may already be fully utilized without the burden of replicating a full Active Directory database.

Windows Server 2003 introduced a new feature called *Install from Media* (IFM) that allows domain controllers to be promoted from a backup of another domain controller in the domain. The only replication that will take place in order for the promotion to complete is the changes that have occurred since the backup was taken. The backup image can be either copied to the domain controller via SMB or another protocol, or shipped out of band using a commercial courier. Shipping the backup this way substantially lessens the burden on the WAN for promotion of the domain controller and in some scenarios can substantially speed up the promotion process.

The process for promoting a domain controller from IFM media varies substantially between Windows Server 2003 and Windows Server 2008 domain controllers. On Windows Server 2008 you must create the IFM media using *ntdsutil* and then point the dcpromo wizard at the folder *ntdsutil* creates. Windows Server 2003 domain controllers require that you restore a system state backup taken with NT Backup to an "alternate location" and then point the dcpromo wizard at this folder.

Creating and using IFM media on Windows Server 2003

Working with IFM on Windows Server 2003 is very similar to doing so on newer versions of Windows, but there are a few key differences:

- You must create the IFM media using a system state backup from NT Backup.
- To use the backup, you must restore it on the target system using the "Restore files to: Alternate location" option shown in Figure 18-7.
- *dcpromo* must be started directly in advanced mode by running *dcpromo /adv*.

Creating and using IFM media on Windows Server 2008 and newer

Creating the IFM media is a straightforward process, as shown in the following example. You will be given the choice of which type of media you'd like to create:

- Full writable domain controller (without Sysvol)
- Full writable domain controller (with Sysvol)
- Read-only domain controller (without Sysvol)
- Read-only domain controller (with Sysvol)

The difference between full domain controller and read-only domain controller media is that the latter do not contain any passwords or other secrets. While you can include the contents of Sysvol, the dcpromo wizard will not use them during promotion to seed the local Sysvol share:

```
ntdsutil: activate instance ntds
Active instance set to "ntds".
```

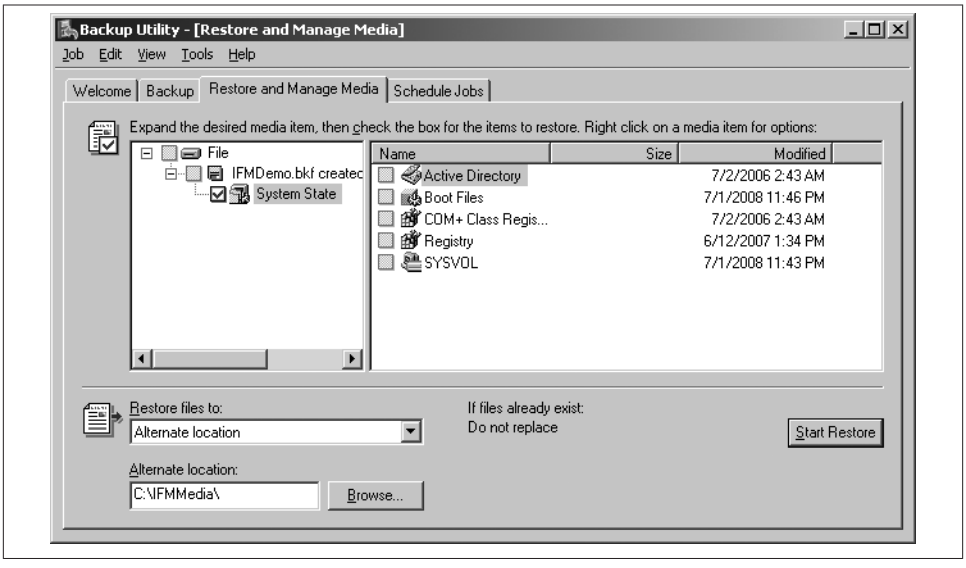


Figure 18-7. Restoring to an alternate location

```
ifm: create full c:\ifmmedia
Creating snapshot...
Snapshot set {f15b79ab-9b0b-4845-9583-eb6c7a49bee9} generated successfully.
Snapshot {b1681552-114b-44b9-b547-9a1b15104907} mounted as C:\$SNAP_200807012230
_VOLUMEC$\
Snapshot {b1681552-114b-44b9-b547-9a1b15104907} is already mounted.
Initiating DEFRAGMENTATION mode...
     Source Database: C:\$SNAP_200807012230_VOLUMEC$\Windows\NTDS\ntds.dit
     Target Database: c:\ifmmedia\Active Directory\ntds.dit

                Defragmentation  Status (% complete)

          0    10   20   30   40   50   60   70   80   90  100
          |----|----|----|----|----|----|----|----|----|----|
          ..................................................

Copying registry files...
Copying c:\ifmmedia\registry\SYSTEM
Copying c:\ifmmedia\registry\SECURITY
Snapshot {b1681552-114b-44b9-b547-9a1b15104907} unmounted.
IFM media created successfully in c:\ifmmedia
```

You can also create IFM media from a system state backup using the Windows Server 2008 R2 and newer versions of Windows Server Backup with the "Alternate location" option shown in Figure 18-16.

In the preceding example, we've created IFM media for promoting a full writable domain controller excluding the contents of the Sysvol. Figure 18-8 shows the contents of *c:\ifmmedia*.

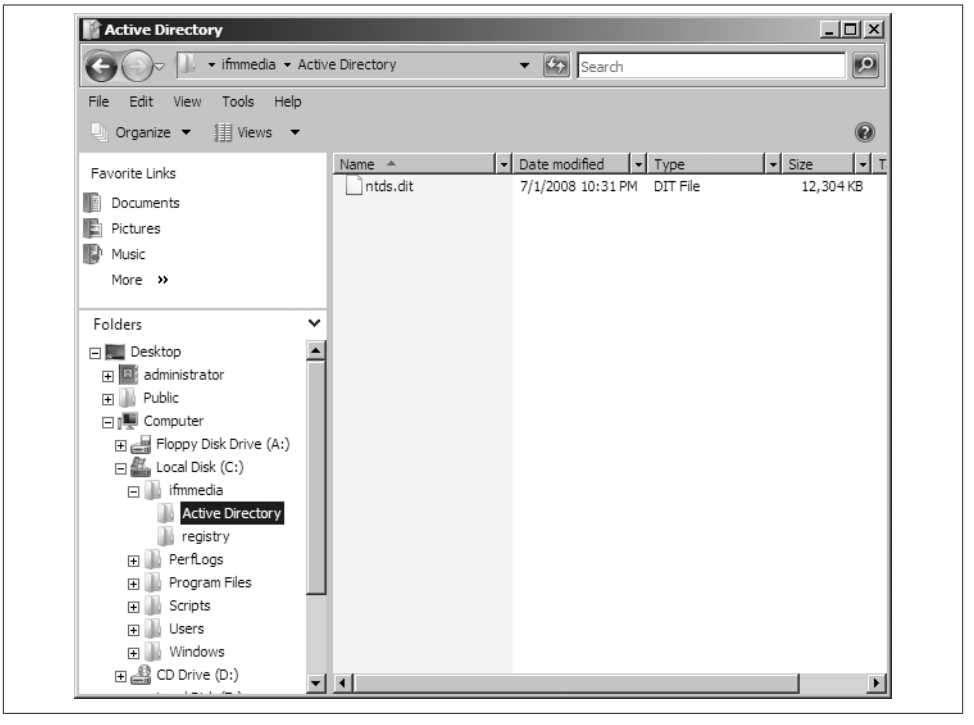


Figure 18-8. Contents of IFM media

You can use IFM media created on a 32-bit domain controller, to promote a 64-bit domain controller, and vice versa. You cannot, however, use RODC media to promote a full domain controller.

Once you have created the IFM media, you can copy it to the new/target domain controller using whatever means are most convenient. In order to utilize the media, you will need to start dcpromo in advanced mode. Advanced mode is accessible by checking the "Use advanced mode installation" checkbox on the first page of the wizard, as shown in Figure 18-9. You can also run *dcpromo /adv*.

Figure 18-9. Starting dcpromo in advanced mode

As you proceed through the wizard, you will be shown a page similar to Figure 18-10 that gives you the option to use the IFM media.

When the wizard completes, it will modify the backup of the *ntds.dit* database from the source domain controller and then replicate any changes made since the backup of the database was created.

Restoring Active Directory

No one ever wants to be in a position where they have to restore Active Directory, but nevertheless you should prepare for the possibility. Good planning and preparation helps to ensure that you will have a minimal number of surprises in the event of a disaster scenario that requires you to perform a restore. Active Directory restores come in a few different flavors, which we'll cover now.

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 to account for 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

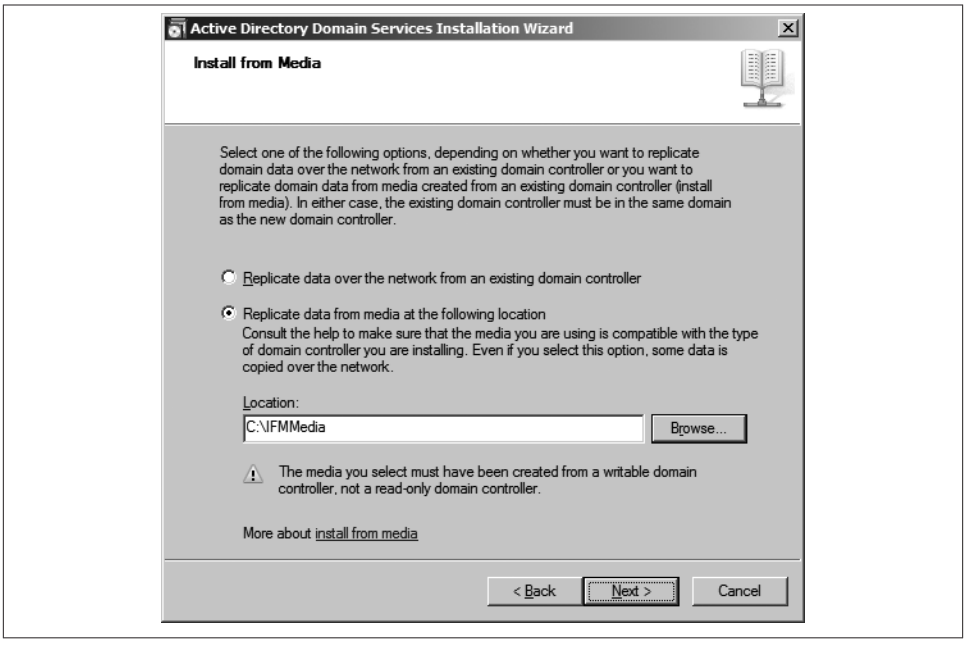


Figure 18-10. Using Install from Media

here is that this process is for machines that are currently functioning properly as domain controllers; there is no need to rebuild from the ground up. There may be some circumstances when you want to perform a similar restore, but the server is already configured as 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 hadn't replicated to another server.

A nonauthoritative restore simply restores Active Directory without marking any of the data as authoritative. This means that 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 nonauthoritative restore of a domain controller, you need to boot the DC into Directory Services Restore Mode (DSRM). 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 *ntds.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 and be unable to continue.

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 18-11.

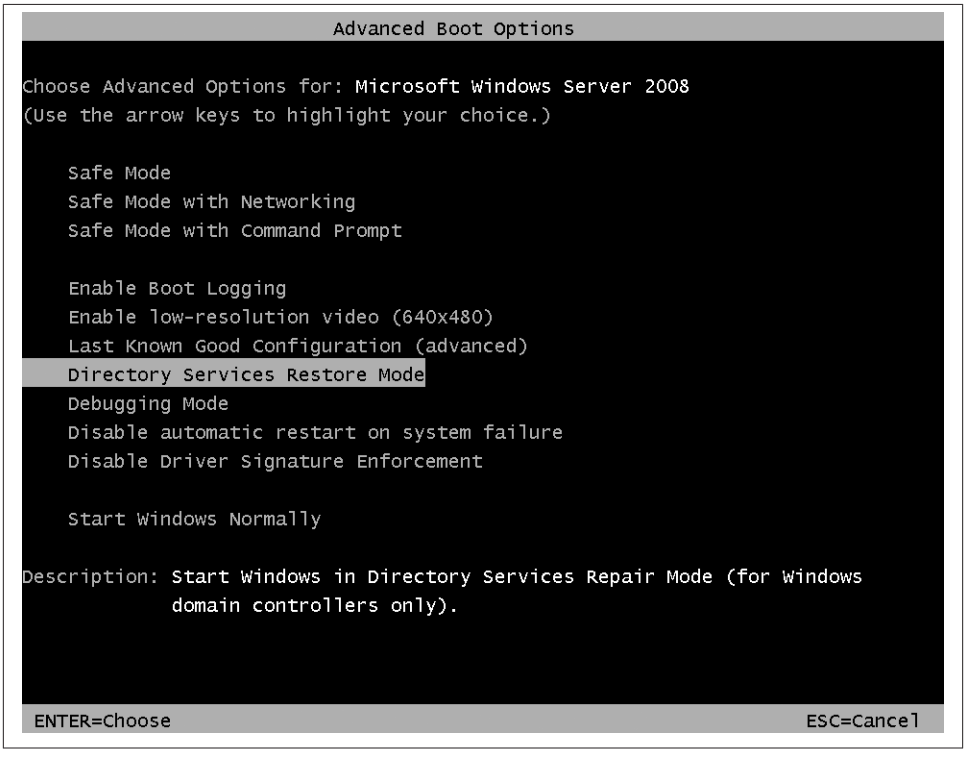


Figure 18-11. Directory Services Restore Mode

Once you receive a logon prompt, you have to log in with the DS Restore Mode administrator account and password. You set the password for this account when you initially *dcpromo* the machine into Active Directory. Since Active Directory is offline in DS Restore Mode, you have to log in with the local administrator account that is stored in the local SAM and that can only be used in this mode. If you do not know the DSRM password, refer to Microsoft Knowledge Base article 322672 for steps to reset the password.

Restoring with NT Backup

After logging into the system, you'll need to bring up the NT Backup utility or other backup software. We will walk through how to do the restore using NT Backup. After clicking Next at the initial wizard screen, you'll see the screen shown in Figure 18-12.

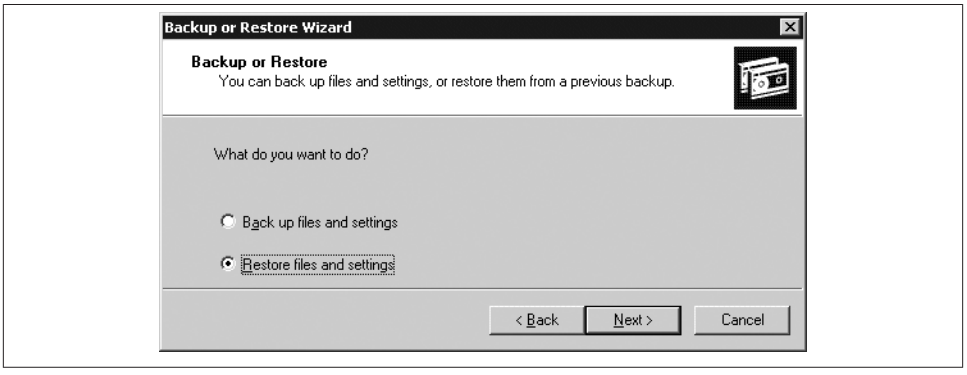


Figure 18-12. Backup or restore options

Select "Restore files and settings" and click Next. You'll now be brought to a screen where you can select what to restore. You should restore at least the system state, but you can also restore the system drive and other drives if necessary. Figure 18-13 shows the selection screen.

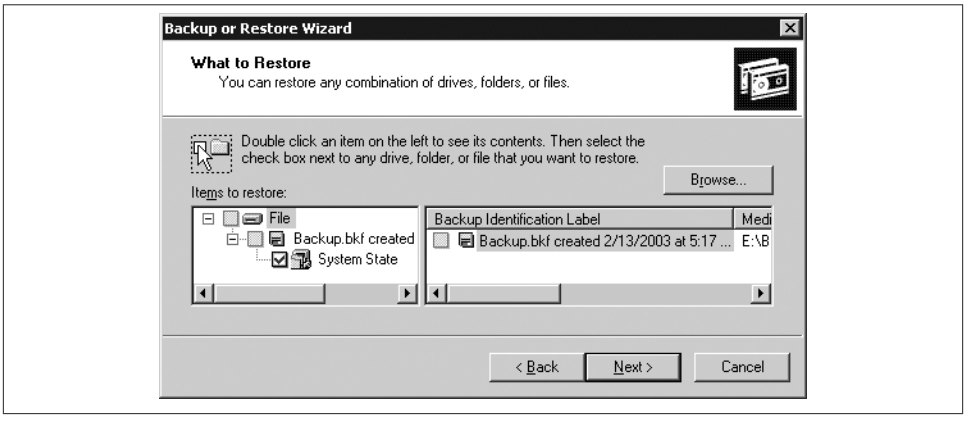


Figure 18-13. Restore selection

After you've made your selection and clicked Next, the summary screen will be displayed, showing what will be restored. Before finishing, you need to click the Advanced

button and walk through the advanced screens to ensure that junction points will be restored, as shown in Figure 18-14.

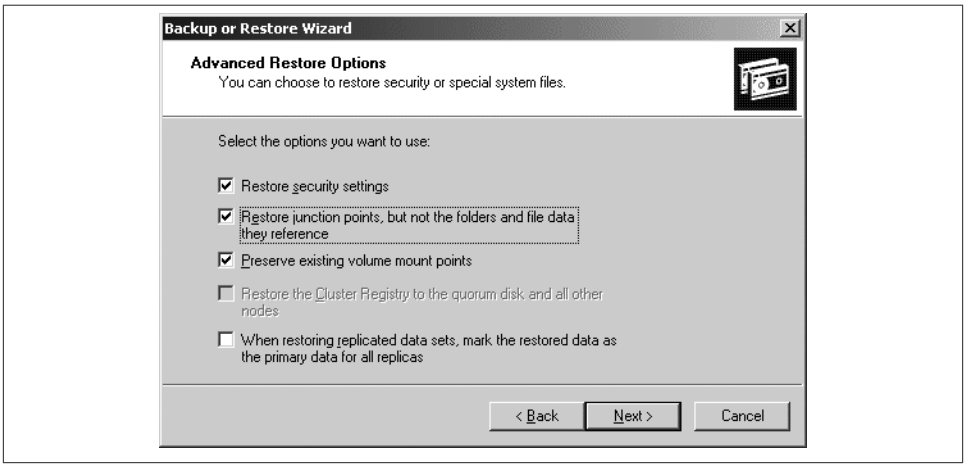


Figure 18-14. Restore junction points

Click Finish to kick off the restore. After the restore is complete, you'll need to reboot into normal mode. At this point, the domain controller will replicate the latest changes with its replication partners. Give time for the replication to complete and then monitor the server and check the event logs to make sure it is functioning correctly.

Restoring with Windows Server Backup

Restoring from a system state backup can be performed using the Recover option in the task pane. First, in the Recovery Wizard, locate the drive or network share containing the backup you will be restoring from. Next, select the date the backup was taken on from the calendar and the time from the drop-down on the right, as shown in Figure 18-15.

If you need to authoritatively restore Sysvol, select the "Perform an authoritative restore of Active Directory files" option. This will only perform an authoritative restore of Sysvol, contrary to the text on the screen, as shown in Figure 18-16.

Once you have completed the wizard, you'll need to reboot into normal mode. At this point, the domain controller will replicate the latest changes with its replication partners. Give time for the replication to complete and then monitor the server and check the event logs to make sure it is functioning correctly. If you need to perform an authoritative restore, continue reading this chapter before rebooting.

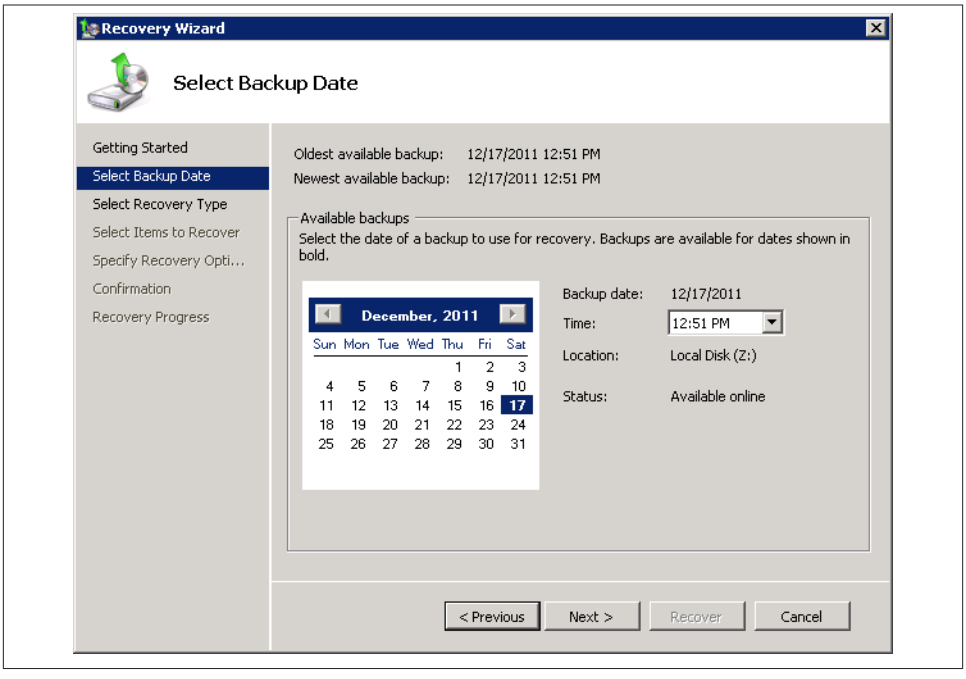


Figure 18-15. Selecting a backup to restore from with Windows Server Backup

Partial Authoritative Restore

In some situations, you may need to restore data in Active Directory. In the examples we've shown so far of restoring a domain controller and performing a nonauthoritative restore, we simply wanted to get the domain controller back up and running. There are certain situations, though, in which you may need to do an authoritative restore. Here are a few examples:

- Accidental deletion of important objects
- Accidental deletion of an entire subtree
- Corruption of objects or the entire directory
- Reversing certain object additions or modifications

In all of these scenarios, you can do a partial authoritative restore to reverse the changes. If the entire directory gets corrupted, you'll need to do a complete authoritative restore, which we will touch on shortly.

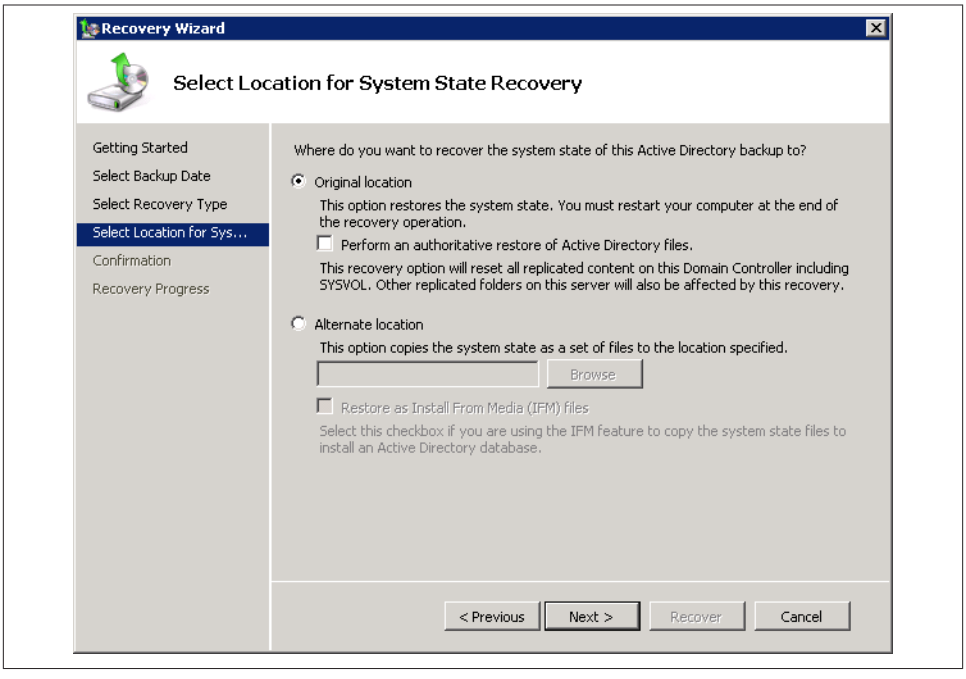
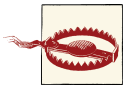


Figure 18-16. System State Recovery location



You cannot authoritatively restore the Schema naming context. In order to roll back the schema, you must rebuild the entire forest either from scratch or from a set of system state backups that predate the schema modification. The feasibility of this option is likely small in almost any environment.

You have two options for doing an authoritative restore. You can either find a domain controller that has the data it is supposed to, perhaps because the changes haven't replicated to it yet, or you can restore the data from a backup. In either case, you need to boot into DS Restore Mode, as described in the section "Nonauthoritative Restore" on page 516. Again, this is necessary due to the fact that the Active Directory database is locked when the DC is live, and no modifications can be made. Once you are in DS Restore Mode, you can restore from a backup if necessary, as described earlier.

At this point you need to mark the data you want restored as authoritative in your offline Active Directory database. This is done with the *ntdsutil* utility. There are several options to choose from under the `authoritative restore` menu shown here:

```
ntdsutil: authoritative restore
authoritative restore: ?
 ?                              - Show this help information
```

```
Create ldif file(s) from %s   - Creates ldif file(s) using specified
                                authoritatively restored objects list
                                to recreate back-links of those objects.
Help                          - Show this help information
List NC CRs                   - Lists Partitions and cross-refs.  You need
                                the cross-ref of a Application Directory
                                Partition to restore it.
Quit                          - Return to the prior menu
Restore database              - Authoritatively restore entire database
Restore database verinc %d    - ... and override version increase
Restore object %s             - Authoritatively restore an object
Restore object %s verinc %d   - ... and override version increase
Restore subtree %s            - Authoritatively restore a subtree
Restore subtree %s verinc %d  - ... and override version increase
authoritative restore:
```

When doing a partial restore, you can use either the *restore object %s* subcommand to restore a single object or the *restore subtree %s* subcommand to restore an entire subtree of objects. In the following example, we will restore the *jsmith* user object:

```
authoritative restore: restore object cn=jsmith,ou=sales,dc=mycorp,dc=com
Opening DIT database... Done.
 The current time is 08-10-05 00:15.25.
Most recent database update occured at 08-09-05 21:48.51.
Increasing attribute version numbers by 100000.
 Counting records that need updating...
Records found: 0000000001
Done.
  Found 1 records to update.
 Updating records...
Records remaining: 0000000000
Done.
Successfully updated 1 records.
 The following text file with a list of authoritatively restored
objects has been created in the current working directory:
        ar_20050810-001525_objects.txt
 One or mofre specified objects have back-links in this domain.
The following LDIF files with link restore operations have
 been created in the current working directory:
        ar_20050810-001525_links_ad.loc.ldf
 Authoritative Restore completed successfully.
authoritative restore: quit
```

As you can see, *ntdsutil* increases the object's version number by 100,000. This is how it is marked as authoritative in the database. After you reboot into normal mode, the domain controller will check with its replication partners and determine that the *jsmith* user object has a higher version than its partners have. It will then replicate this out to them, and likewise, all other objects that have been updated on the partner will be replicated to this server.

If, for whatever reason, the auto increment of 100,000 is not enough for the object(s), you can use the alternate subcommand *restoreobject %s verinc %d*, where *%d*, is the amount to increment the version by (this figure is multiplied by the number of days since the backup).

Depending on the version of *ntdsutil*, you might see different results than what you see here. The utility had some major changes incorporated in the Windows Server 2003 SP1 version; for example, it now creates an LDF file that will contain any linked attribute values attached to the object in the same domain, which allows for recovery of group membership and other linked attributes.

Complete Authoritative Restore

Restoring the entire Active Directory database is similar in concept to restoring individual objects or subtrees, except you are restoring all of the objects, with the exception of the schema. This should be done with caution and only in the most extreme situations. We highly recommend that you test this out in a lab environment to ensure you have the process correctly documented and you actually have experience with performing restores.

Again, to run the restore command, you have to be in DS Restore Mode, and you need to have restored the system from a backup, as described previously in this chapter. The following is example output from the *restore database* subcommand:

```
authoritative restore: restore database
Opening DIT database... Done.
  The current time is 08-10-05 00:39.46.
Most recent database update occured at 08-09-05 21:48.51.
Increasing attribute version numbers by 100000.
 Counting records that need updating...
Records found: 0000001725
Done.
  Found 1725 records to update.
 Updating records...
Records remaining: 0000000000
Done.
  Successfully updated 1725 records.
 The following text file with a list of authoritatively
restored objects has been created in the current working directory:
        ar_20050810-003946_objects.txt
 One or more specified objects have back-links in this domain.
The following LDIF files with link restore operations have
 been created in the current working directory:
        ar_20050810-003946_links_ad.loc-Configuration.ldf
        ar_20050810-003946_links_ad.loc.ldf
 Authoritative Restore completed successfully.
 authoritative restore: quit
```

If you have to perform a complete authoritative restore, the assumption is that something catastrophic has happened on a domain controller that caused some form of global, irreparable Active Directory corruption. The safest thing may in fact be to restore one domain controller per domain and rebuild the rest. You would need to manually remove each of the rebuilt domain controllers from Active Directory and then repromote each. Again, this is only a suggestion, and each situation must be thoroughly thought out before taking such drastic measures. If you have a thorough understanding of the processes and procedures involved and practice them, you will certainly be better prepared to tackle such an eventuality. For the Microsoft documentation on Active Directory forest recovery best practices, see this website.

Working with Snapshots

An Active Directory feature introduced in Windows Server 2008 was the notion of *snapshots*. With snapshots, you can make a point-in-time copy of the Active Directory database. This feature uses the Volume Shadow Copy (VSS) service to create the snapshot. Once you have created a snapshot, you can mount it using the *dsamain* command-line utility and browse it like the live Active Directory database using *ldp*, Active Directory Users and Computers (ADUC), or any other tool that supports the LDAP protocol.

The benefits of this new capability are endless, but some ideas include:

- Looking at permissions or other settings prior to modification in case you need to roll back
- Exporting objects or values that were inadvertently modified and then importing them back into your production directory
- Determining which backup to perform a restore from
- Restoring a complete object with a third-party tool

Creating the snapshot is a straightforward process using the *ntdsutil* command-line utility:

```
C:\> ntdsutil
ntdsutil: snapshot
snapshot: activate instance ntds
Active instance set to "ntds".
snapshot: create
Creating snapshot...
Snapshot set {e73b71cd-7e2b-40ee-8871-69575f4b1e66} generated successfully.
```

Once you have created a snapshot, you can mount it and make it available to LDAP clients. Mounting the snapshot consists of either specifying the GUID generated in the previous step or selecting the snapshot from a list. In this example, we select the snapshot from a list:

```
snapshot: list all
 1: 2008/06/30:22:44 {e73b71cd-7e2b-40ee-8871-69575f4b1e66}
 2:   C: {c8bc59c0-f20b-4bac-a12a-ab6eab737f0c}

snapshot: mount 1
Snapshot {c8bc59c0-f20b-4bac-a12a-ab6eab737f0c} mounted as
C:\$SNAP_200806302244_VOLUMEC$\
```

Once the snapshot has been mounted in the filesystem, you can use the *dsamain* utility to mount the snapshot version of the Active Directory database and make it accessible via LDAP. *dsamain* requires that you provide the path to the database to mount as well as a port number to listen on. If you only specify the LDAP port, *dsamain* will automatically use the subsequent three ports for LDAP over SSL (LDAPS), Global Catalog, and Global Catalog SSL connections. Thus, if you specify port 10389 for LDAP, ports 10390, 10391, and 10392 will also be used. The command to mount the database in this scenario is:

```
dsamain -dbpath C:\$SNAP_200806302244_VOLUMEC$\Windows\NTDS\ntds.dit
-ldapport 10389
```

You can then use LDP or any LDAP client to connect to port 10389, as shown in Figure 18-17.

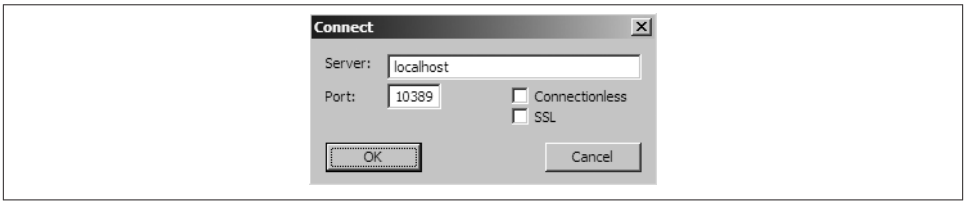


Figure 18-17. Connecting to a mounted snapshot

You can also connect to mounted snapshots using Active Directory Users and Computers:

1. Launch ADUC.
2. Right-click the domain and select Change Domain Controller
3. Select "This Domain Controller or AD LDS instance."
4. Click "<Type a Directory Server name[:port] here>."
5. Enter the name of a domain controller or the localhost, as shown in Figure 18-18, and click OK.

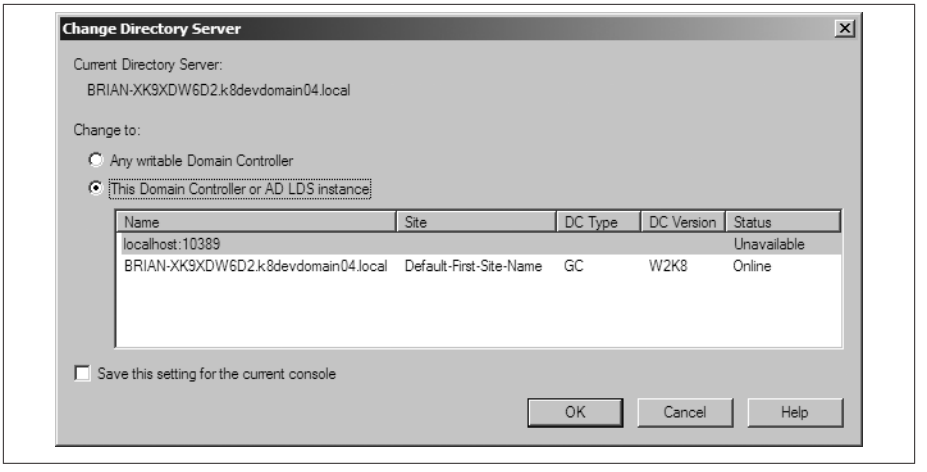


Figure 18-18. Connecting to a mounted snapshot with ADUC

Once you're done with a snapshot, you need to stop the *dsamain* instance and unmount the snapshot. To stop the *dsamain* instance, simply press Ctrl-C in the command window and it will exit. Unmounting the snapshot is similar to mounting it:

```
C:\> ntdsutil
ntdsutil: snapshot
snapshot: list all
 1: 2008/06/30:22:44 {e73b71cd-7e2b-40ee-8871-69575f4b1e66}
 2:   C: {c8bc59c0-f20b-4bac-a12a-ab6eab737f0c} C:\$SNAP_200806302244_VOLUMEC$\

snapshot: unmount 1
Snapshot {c8bc59c0-f20b-4bac-a12a-ab6eab737f0c} unmounted.
```

In summary, Active Directory snapshots allow you to take a point-in-time copy of the Active Directory database on a domain controller and mount it for parallel access via LDAP. For more information on this functionality, check out the guide at this website.

Active Directory Recycle Bin

One of the most common Active Directory disasters that administrators face is the accidentally deleted object, or more commonly, accidentally deleted objects. While this type of disaster is entirely preventable, it comes up time and time again. Various third parties have long made tools available for purchase that enable you to restore objects or attribute values to the directory without recovering from a conventional system state backup. Windows Server 2008 R2 introduced the Active Directory Recycle Bin, which is Microsoft's solution to recovering deleted objects.

Prior to the advent of the Active Directory Recycle Bin, one of the side effects of deleting an object was that the majority of its attributes were removed, so an undelete or

reanimation of the tombstone object would be missing most attributes that were set when the object was deleted. Only those attributes that were marked in the schema to be preserved on a tombstone were retained. While most attribute values can be retained, linked attributes (such as group membership) cannot be retained on a tombstone.

Deleted Object Lifecycle

When you delete an object from Active Directory, a number of things happen. Chiefly, the object isn't actually purged from the Active Directory database for quite some time, in order to ensure that replication of the deletion occurs across the forest. Objects in this state are known as *tombstones*. If the Active Directory Recycle Bin is not enabled, the time that objects are retained in the database after deletion is called the *tombstone lifetime*. Once the tombstone lifetime passes, the garbage collection process on each domain controller removes the records from the database. Figure 18-19 shows the lifecycle of an object that is deleted from Active Directory when the Recycle Bin is not enabled.

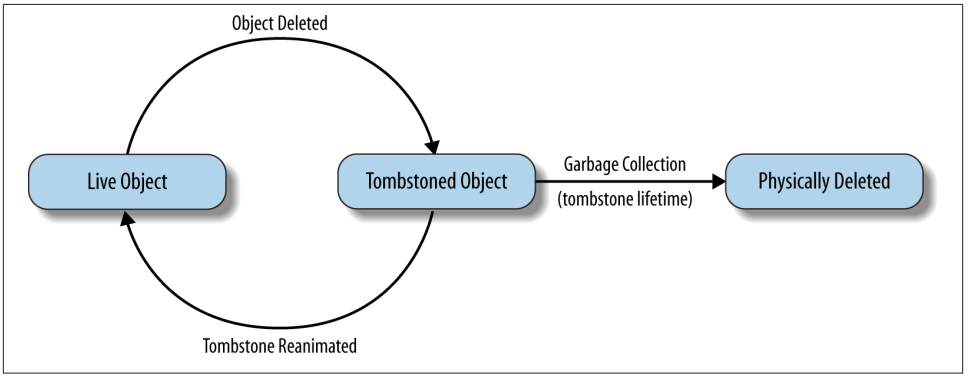


Figure 18-19. Deleted object lifecycle (without Active Directory Recycle Bin)

When you enable the Active Directory Recycle Bin, before a deleted object becomes a tombstone it first enters the *deleted object state*, as shown in Figure 18-20. When an object is in the deleted state, all of its attributes are retained, as well as any links to the object. Similar to how an object was deleted prior to the Recycle Bin being enabled, the `isDeleted` attribute is set to `TRUE`, and the object is moved to the Deleted Objects container.

By default, garbage collection runs on each domain controller every 12 hours. During garbage collection, tombstones are checked to see if the time at which they were deleted exceeds the tombstone lifetime. If so, the records are removed.

You can trigger garbage collection to run on demand by setting the `doGarbageCollection` operational attribute to `1` on a given domain controller using a tool such as *ldp* or *admod*. For example, you can use the following *admod* command to run garbage collection on server *DC01*:

```
admod -h dc01 -sc gc
```

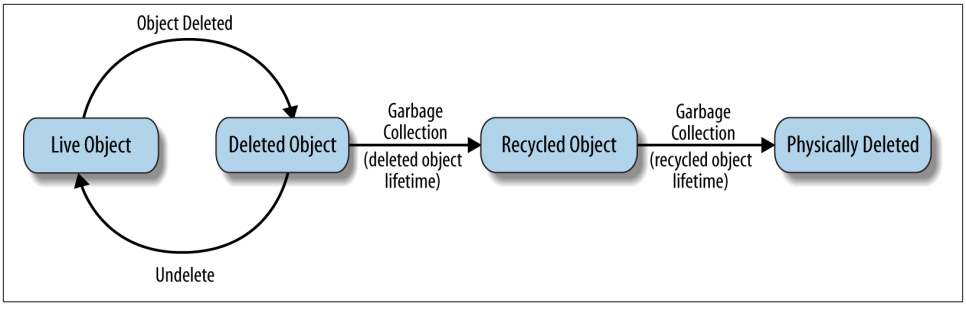


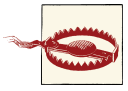
Figure 18-20. Deleted object lifecycle with Active Directory Recycle Bin enabled

After the deleted object lifetime passes, garbage collection converts the object to a recycled object, strips all of the attributes that are not marked for preservation on tombstones, and sets the `isRecycled` attribute to `TRUE`. The recycled object state is what was previously called a tombstone. Finally, once the recycled object lifetime (previously called the tombstone lifetime) passes, garbage collection physically removes the record from the database.

Unlike tombstones, recycled objects cannot be reanimated. In order to recover an object once it enters the recycled state, you must authoritatively restore the object from a backup taken while the object was still in the deleted state.

Enabling the Recycle Bin

The Active Directory Recycle Bin requires the Windows Server 2008 R2 forest functional level (level 4) or better. Once the forest functional level requirement is met, the Recycle Bin optional feature can be enabled. Enabling the Active Directory Recycle Bin is most easily done with PowerShell.



Once you have enabled the Active Directory Recycle Bin, it cannot be disabled. Be sure that you have tested the effects of enabling the Recycle Bin in a test environment including tools that integrate with Active Directory. Tools generally affected by enabling the Recycle Bin include those that use LDAP DirSync to synchronize changes in AD with a local database.

The Microsoft Identity Lifecycle Manager (ILM) and some versions of Office365 DirSync are two examples of applications that are affected by the Recycle Bin.

On a machine with the Active Directory module for Windows PowerShell installed, run the following two commands:

```
Enable-ADOptionalFeature -Identity 'CN=Recycle Bin Feature,
CN=Optional Features,CN=Directory Service,CN=Windows NT,
CN=Services,CN=Configuration,'<Forest Root DN>'
-Scope ForestOrConfigurationSet
-Target '<Forest DNS Name>' `'CN=Recycle Bin Feature, ` CN=Optional Features, `
CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,
<Forest Root DN>' -Scope ForestOrConfigurationSet -Target '<Forest DNS Name>'

WARNING: Enabling 'Recycle Bin Feature' on 'CN=Partitions,CN=Configuration,
<Forest Root DN>' is anirreversible action! You will not be able to disable
'Recycle Bin Feature' on 'CN=Partitions,CN=Configuration,<Forest Root DN>'
if you proceed.
```

If you were enabling the Recycle Bin for the *contoso.com* forest, you would replace the <*Forest Root DN*> in the sample with `DC=contoso,DC=com`, and you would replace the <*Forest DNS Name*> with `contoso.com`.

You can also enable the Active Directory Recycle Bin using the Windows Server 2012 version of the Active Directory Administrative Center. Simply right-click the forest root domain and click Enable Recycle Bin.

Once the Recycle Bin is enabled, you may see some additional replication traffic in your environment as the `isRecycled` attribute is updated for each object in the forest. Additionally, objects that were previously tombstoned will enter the recycled state and will no longer be available for reanimation. Finally, once the Recycle Bin is enabled, the infrastructure master FSMO role no longer has any work to perform, so its placement is no longer important. For more information on FSMO roles, refer to Chapter 2.

Undeleting Objects

Recovering objects from the Recycle Bin, or undeleting them, can be a tricky process. If you are recovering an entire tree of objects, such as in the case that an OU has been deleted, you need to undelete the objects in the correct order. That is, it's not possible to recover an object before its parent has been recovered.

Fortunately, Microsoft provides a number of tools to make this easier. If you're using the Windows 8 Remote Server Administration Tools (RSAT), you can use the Active Directory Administrative Center (ADAC) to recover. With the Windows Server 2008 R2 RSAT tools, or if you need more granularity, you can use PowerShell.

Using ADAC

Recovering individual objects from the recycle bin with ADAC is a very straight-forward process. Once the recycle bin is enabled, you will see a Deleted Objects container in ADAC under the root of the domain. Find the object you want to recover and right-click it. You can either choose Restore to restore the object to its original location, or, if you want to restore the object to a different OU, choose "Restore to." The downside to using ADAC is that it does not support the notion of tree recovery. If you need to recover an OU hierarchy, you'll need to resort to PowerShell and a special tree recovery script discussed in the next section.

Using PowerShell

The cmdlet you'll need to become most familiar with when recovering objects from the Recycle Bin with PowerShell is *Restore-ADObject*. In order to recover an object or series of objects, you'll first need to identify them and provide them to the *Restore-ADObject* cmdlet. You can either provide the DN (or GUID) of a specific object to restore, or pipe the results of the *Get-ADObject* cmdlet to *Restore-ADObject*.

The easiest way to become familiar with these options is with a few examples. In the first example, we will recover the account for user *gwashington*. We are able to locate this user based on the previous relative distinguished name (RDN). In the case of a user (or computer), the RDN is the `CN=` component of the object's DN. In the case of an organizational unit, the RDN is the `OU=` component of the object's DN. We recover the user object as follows:

```
Get-ADObject -IncludeDeletedObjects -LdapFilter `
"(&(objectClass=user)(msDS-lastKnownRDN=gwashington))" | Restore-ADObject
```

Consider another example where you accidentally delete an OU containing a number of objects. In this case, you'll first need to recover the OU and then recover the objects that were inside of it. In this example we'll recover an OU called Presidents, as shown in Figure 18-21, and then recover all of the objects that were in the OU.

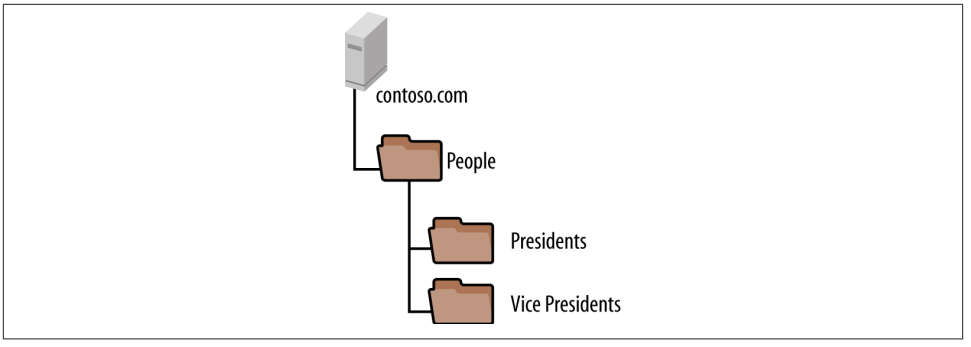


Figure 18-21. Hierarchy for object recovery

In order to recover the deleted Presidents OU and all of the objects in it, we use the following PowerShell commands (each on one line):

```
Get-ADObject -IncludeDeletedObjects -LdapFilter ` "(&(objectClass=organizationalUnit)
(msDS-lastKnownRDN=Presidents))" `
| Restore-ADObject

Get-ADObject -IncludeDeletedObjects -LdapFilter `
"(&(objectClass=user)(lastKnownParent=OU=Presidents,
OU=People,DC=contoso,DC=com))"
| Restore-ADObject
```

If you were to try to restore an object before restoring the object's parent, you would receive an error similar to the following:

```
Restore-ADObject : The operation could not be performed because the object's
parent is either uninstantiated or deleted
```

Fortunately, the *Restore-ADObject* cmdlet has a number of parameters, as shown in Table 18-2, that enable you to modify the cmdlet's default behavior.

Table 18-2. Restore-ADObject parameters

Parameter	Description
Identity	Specifies the object to restore. You can provide the DN of the deleted object (in the Deleted Objects container), or the object's GUID.
NewName	Specifies a new RDN for the object to use when it is restored. If this is not provided, the `msDS-lastKnownRDN` value is used instead. You may find this parameter useful if, for example, you are recovering a deleted OU after someone has manually recreated the OU with the same name.
TargetPath	Specifies where in the directory to recover the object to. If the object's parent no longer exists or you want to recover the object to a new location, you should provide the DN of that location here. If a value is not provided, the recovered object's `lastKnownParent` value is used instead.

The examples we've looked at so far are trivial in comparison to possible accidental deletion scenarios where a complex hierarchy of OUs and objects may have been deleted. Such a hierarchy will have to be restored from the top down. Doing this manually would be complex and time-consuming at best. Fortunately, Microsoft has provided a script, *Restore-ADTree.ps1*, that can be used to recover a hierarchy of objects. This script is available at this link.

If, for example, the People OU shown in Figure 18-21 were accidentally deleted, you could use *Restore-ADTree.ps1* to recover the hierarchy like this:

```
Restore-ADTree.ps1 -lastKnownRDN "People" -lastKnownParent "dc=contoso,dc=com"
```

If you receive an error similar to the following when you run the script, you need to modify your PowerShell Execution Policy.

```
File C:\Restore-ADTree.ps1 cannot be loaded because the execution of
scripts is disabled on this system. Please see
"get-help about_signing" for more details.
```

To set your execution policy to `RemoteSigned`, run this command:

```
Set-ExecutionPolicy -ExecutionPolicy:RemoteSigned
```

For more information about execution policies, refer to this website.

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 regarding how to handle FSMO failures.

While it would be a really nice feature if domain controllers could detect that they were being shut down and gracefully transfer any FSMO roles they held other domain controllers, this isn't how it works in reality. In the absence of graceful FSMO role transfer, you have to perform manual transfers. To manually transfer a FSMO role, 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-ins that can be used to transfer each 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

You can use the command-line tool *netdom* to query all FSMOs for a given domain with a single command:

```
netdom query fsmo /domain:domainname
```

When a FSMO role owner goes down and cannot be brought back online, you can no longer transfer the role; you instead have to "seize" it. Windows Server 2008 and newer domain controllers will automatically seize any FSMO roles held by a domain controller when you perform a metadata cleanup. If you wish to specify exactly where the roles are seized to you must still perform this task by hand, though. Unfortunately, you cannot seize FSMO roles using the Active Directory snap-ins, as you can to transfer them. To seize a FSMO role, you will need to use the *ntdsutil* utility. 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 start off by getting into the *ntdsutil* interactive mode and looking at the options for the *roles* command:

```
C:\> ntdsutil
ntdsutil: roles
fsmo maintenance: ?
 ?                              - Show this help information
 Connections                    - Connect to a specific domain controller
 Help                           - Show this help information
 Quit                           - Return to the prior menu
 Seize domain naming master     - Overwrite domain role on connected server
 Seize infrastructure master    - Overwrite infrastructure role on connected
                                  server
 Seize PDC                      - Overwrite PDC role on connected server
 Seize RID master               - Overwrite RID role on connected server
 Seize schema master            - Overwrite schema role on connected server
 Select operation target        - Select sites, servers, domains, roles and
                                  naming contexts
 Transfer domain naming master - Make connected server the domain naming master
 Transfer infrastructure master - Make connected server the infrastructure master
 Transfer PDC                   - Make connected server the PDC
 Transfer RID master            - Make connected server the RID master
 Transfer schema master         - Make connected server the schema master
```

We must now connect to the domain controller to which we want to seize the role. In this case, we will connect to *DC1*:

```
fsmo maintenance: connections
server connections: connect to server dc1
```

```
Binding to dc1 ...
Connected to dc1 using credentials of locally logged on user.
server connections: quit
```

At this point, we can transfer and seize any available FSMO role to the *DC1* domain controller. In the next example, we will attempt to seize the schema master role. The current schema master is *DC2*. If we tried to perform a seizure and *DC2* was operational, we would effectively do a graceful transfer of the role to *DC1*. If *DC2* was not operational, a seizure would take place, as shown in the following output (note that some lines may wrap due to their length):

```
fsmo maintenance: seize schema master
Attempting safe transfer of schema FSMO before seizure.
ldap_modify_sW error 0x34(52 (Unavailable).
Ldap extended error message is 000020AF: SvcErr: DSID-03210300, problem 5002
(UNAVAILABLE), data 1753
 Win32 error returned is 0x20af(The requested FSMO operation failed. The current
FSMO holder could not be contacted.)

Depending on the error code this may indicate a connection,
ldap, or role transfer error.
Transfer of schema FSMO failed, proceeding with seizure ...
Server "dc1" knows about 5 roles
Schema - CN=NTDS Settings,
CN=DC1,CN=Servers,CN=RTP,CN=Sites,
CN=Configuration,
DC=mycorp,DC=com
Domain - CN=NTDS Settings,CN=DC1,CN=Servers,CN=RTP,CN=Configuration,
DC=mycorp,DC=com
PDC - CN=NTDS Settings,CN=DC1,CN=Servers,CN=RTP,CN=Sites,CN=Configuration,
DC=mycorp,DC=com
RID - CN=NTDS Settings,CN=DC1,CN=Servers,CN=RTP,CN=Sites,CN=Configuration,
DC=mycorp,DC=com
Infrastructure - CN=NTDS Settings,CN=DC1,CN=Servers,CN=RTP,CN=Sites,
CN=Configuration,DC=mycorp,DC=com
```

Note that a connection is first attempted to the current role owner, and if it cannot be reached, *ntdsutil* does the seizure.

One of the nice features of the quirky *ntdsutil* command is that it can be run in interactive mode, as we just showed, or it can be run from a single command line. To accomplish the same seizure using a single command line, we could use:

```
C:\> ntdsutil roles conn "connect to server dc1" q "seize schema master" q q
```

Depending on your needs, you could write a batch script to prompt for the role you want to seize and the DC to transfer or seize the role to. This could help when it gets down to crunch time and you need to seize the role quickly, and you do not want to have to thumb through this book trying to find all of the commands.

Restartable Directory Service

Windows Server 2008 introduced the ability to start and stop Active Directory like a normal Windows service. This allows you to perform most offline operations without restarting the domain controller. While Active Directory is stopped, it will not respond to logon requests. If the domain controller is hosting Active Directory–integrated DNS zones, it will also not respond to queries for these zones. While the Active Directory service is stopped, you can perform all of the offline tasks outlined in this chapter with the exception of restoring from a backup. Restoring still requires that you boot into Directory Services Restore Mode.

Once you have stopped the Active Directory service, you can log into the domain controller with domain credentials if another domain controller is available to service the request. If another domain controller is not available to service the request, you will not be able to log in. If you want to have the option of using the Directory Services Restore Mode password, you must modify the registry. This configuration change is outlined in the sidebar "Modify the DSRM Logon Behavior," next.

Modifying the DSRM Logon Behavior

By default, you must use a domain account to log into a domain controller on which the Active Directory service is stopped.

To change this behavior, you need to create or modify the `REG_DWORD DSRMAdminLogonBehavior` registry value under `HKLM\System\CurrentControlSet\Control\Lsa`.

There are three possible values for this registry value:

Table 18-3.

Value	Description
0	You can only log into the domain controller with a domain account, which requires that another domain controller be available to service the request. This is the default setting.
1	You can log into the domain controller with a domain account or the Directory Services Restore Mode account only when the Active Directory service is stopped.
2	You can log into the domain controller with the Directory Services Restore Mode account regardless of whether the Active Directory service is stopped or started.

If you only have one domain controller in the domain or expect there will be situations where you will not be able to contact another domain controller, you must set this registry value to `1` or you will not be able to log into a domain controller in DS Restore Mode. In general, we recommend that you use a value of `1` for all deployments.

If you are trying to log into a Windows Server 2008 domain controller where the directory service is stopped, and login is failing when you try to use the DSRM credentials, make sure that you are not attempting to log into the domain. If you need to explicitly specify the local administrator account, click Switch User on the logon screen and specify a username of *.\administrator*.

To stop the Active Directory service, open the Services MMC snap-in by going to Start→Run→**`services.msc`**, right-clicking Active Directory Domain Services, and selecting Stop. You will be prompted to also stop dependent services, as shown in Figure 18-22. You can also stop Active Directory from the command prompt by running *net stop ntds*.

Figure 18-22. Stopping dependent services

Once you have restarted the Active Directory Domain Services service, you will need to manually start each of these dependent services.

DIT Maintenance

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 directory information tree. 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 periodically. You may even need to do this more frequently, depending on whether you have people who leave your team and should no longer know the password.

Unfortunately, to accomplish all these tasks (with the exception of changing the DS Restore Mode administrator password) on Windows Server 2003 and earlier domain

controllers, you have to boot the domain controller into DS Restore Mode. That means you will have to 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 an HP Integrated Lights-Out (iLO) connection.

If you cannot access the machine's physical console, you can also connect via Remote Desktop if you modify the DC's boot configuration. If you need to configure a Windows Server 2008 or newer domain controller to automatically boot into DS Restore Mode, you'll need to use the *bcdedit* command-line utility to reconfigure the domain controller's boot settings. This is a three-step process. First, issue this command:

bcdedit /copy {current} /d "DC-DS Restore Mode"

The tool will output a GUID response similar to this:

```
The entry was successfully copied to {49fa7976-5065-11dd-ae21-000c291a8e6c}
```

bcdedit /set *{49fa7976-5065-11dd-ae21-000c291a8e6c}* safeboot dsrepair

bcdedit /default *{49fa7976-5065-11dd-ae21-000c291a8e6c}*

When you restart the domain controller, it will boot into DS Restore Mode. You will then need change the domain controller's default boot option back to the original setting. This is a two-step process. First, issue this command:

`bcdedit /enum`

This will output a list of all the boot manager options. Find the GUID of the normal Windows instance, and issue the following command:

`bcdedit /default` *<guid of normal Windows instance>*

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. While the speed varies greatly based on a number of factors, we've seen some estimates that state it can check around 2 gigabytes per hour, so allocate your change notification accordingly.

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 must restore the database from a backup.

If the integrity check succeeds, you should then run a semantics check. Whereas the integrity check examines the database as a whole, the semantics check will examine the database to determine whether it is healthy as it pertains to Active Directory semantics. Some of the things the semantics check looks at include security descriptors, reference counts, distinguished name tag (DNT) consistency, and deleted objects.

To start a semantics check, run the *go* subcommand from the `semantic database analysis` menu:

```
ntdsutil: semantic database analysis
semantic checker: ?
  ?                           - Show this help information
 Check Quota                  - Integrity-check quota-tracking table
 Get %d                       - Get record info with given DNT
 Go                           - Start Semantic Checker with No Fixup
 Go Fixup                     - Start Semantic Checker with Fixup
 Help                         - Show this help information
 Quit                         - Return to the prior menu
 Rebuild Quota                - Force asynchronous rebuild of quota-tracking
 Verbose %s                   - Turn verbose mode on/off
 semantic checker: go
Fixup mode is turned off
Opening database [Current].......Done.
 Getting record count...3115 records
Getting security descriptor count...82 security descriptors
 Writing summary into log file dsdit.dmp.0
SDs scanned:             82
Records scanned:       3115
Processing records..Done.
 semantic checker: quit
```

If any errors are reported, you can then run *go fixup*, which will attempt to repair any problems.

If you have to run the *repair* or *go fixup* commands, after you boot back into normal mode you should perform a backup as soon as possible and be sure to indicate on the backup that a repair was performed. If, for some reason, you need to restore the domain controller at a later point, and if you restore from a backup prior to the repair, you'll need to perform the same commands to fix the database again. Alternatively, if you start experiencing problems immediately after the repair, you'll want to be able to tell where the last backup was before the repair occurred and restore that copy.

Reclaiming Space

If your domain controllers are running low on disk space or if you have deleted a lot of objects since you promoted your domain controllers, you may want to perform an offline defragmentation of the DIT file. You've probably seen the online defragmentation events that get logged to the Directory Services event log. These include event 700, which states that an online defrag is about to begin, and event 701, which states that the online defrag has completed. The online defrag process runs twice a day by default and consolidates free space within the DIT file. It does not reclaim any disk space used by the DIT file, though. To do that, you must perform an offline defragmentation. For information on determining how much free space is available in the DIT, see the upcoming sidebar "Configuring Logging of Available Database Space" on page 541.

If you perform an in-place upgrade of Windows 2000 domain controllers to Windows Server 2003, your *ntds.dit* may develop a substantial amount of free space in it. This is because of an improvement in Windows Server 2003 that stores unique security descriptors once in the entire database rather than each time they are used. Some Active Directory administrators have seen as much as a 40% reduction in the size of their Active Directory database files following this upgrade. If you want to reclaim this space on disk, you will need to perform an offline defragmentation of the *ntds.dit* file.

An offline defragmentation must be done while the domain controller is in Directory Services Restore Mode. You can then use the *ntdsutil* command to compact—that is, defrag—the *ntds.dit* file. This process actually creates a copy of the *ntds.dit* file in an

alternate location. You can then overwrite the existing DIT file with the new compacted version.

Configuring Logging of Available Database Space

If you are wondering how much free space exists in *ntds.dit*, you can set a registry value that will cause Active Directory to log an additional event—event 1646—when an online defragmentation pass is complete.

Figure 18-23 shows this event on a Windows Server 2008 domain controller. In this scenario, the Active Directory database is 12 MB in size and has 1 MB of free space in the database file. You can use this information to help decide if there will be any benefit to performing an offline defragmentation of the database.

To enable this logging, set the `REG_DWORD` value `6 Garbage Collection` under `HKLM\System\CurrentControlSet\Services\NTDS\Diagnostics` to `1`. You do not need to reboot for this change to take effect.

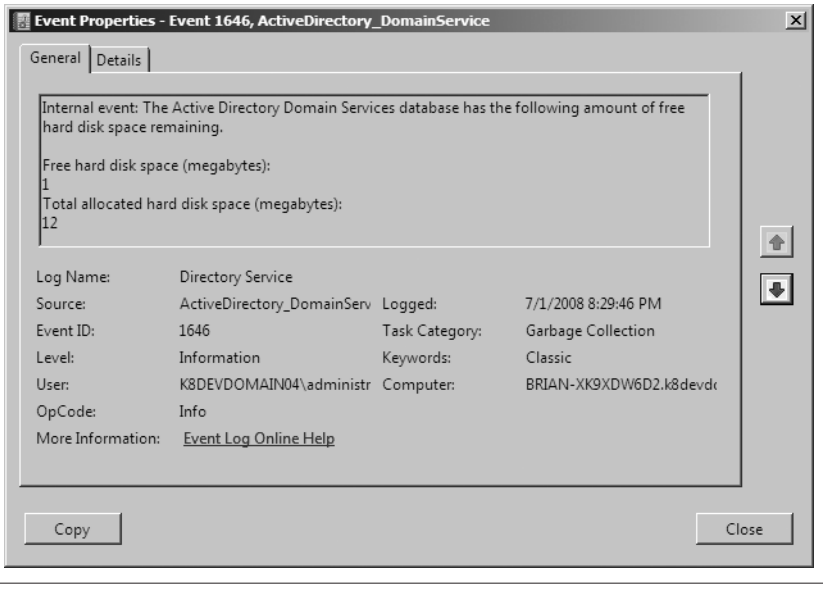


Figure 18-23. Available space in the database

The following shows how to perform an offline defragmentation using *ntdsutil*. After you enter the `files` menu, you'll need to issue the *compact to* <*directorypath*> command. The <*directorypath*> should be the directory in which the new compacted

ntds.dit file will be created. If the directory does not exist, it will be created automatically. You perform the defragmentation as follows:

```
ntdsutil: files
file maintenance: compact to c:\windows\ntds\compact
Opening database [Current].
Creating dir: c:\windows\ntds\compact
Executing Command: C:\WINDOWS\system32\esentutl.exe /d"C:\WINDOWS\NTDS\ntds.dit"
 /t"c:\windows\ntds\compact\ntds.dit" /p /o
Initiating DEFRAGMENTATION mode...
            Database: C:\WINDOWS\NTDS\ntds.dit
      Temp. Database: c:\windows\ntds\compact\ntds.dit
                  Defragmentation Status (% complete)
          0    10   20   30   40   50   60   70   80   90  100
          |----|----|----|----|----|----|----|----|----|----|
          ..................................................
Note:
  It is recommended that you immediately perform a full backup
  of this database. If you restore a backup made before the
  defragmentation, the database will be rolled back to the state
  it was in at the time of that backup.
Operation completed successfully in 20.961 seconds.
Spawned Process Exit code 0x0(0)
If compaction was successful you need to:
   copy "c:\windows\ntds\compact\ntds.dit" "C:\WINDOWS\NTDS\ntds.dit"
and delete the old log files:
   del C:\WINDOWS\NTDS\*.log
file maintenance: quit
```

After you've completed the compaction, you can then decide whether you want to overwrite your current *ntds.dit* file.

Performing an offline defrag of a domain controller affects only that DC. To reclaim space on your other domain controllers, you'll need to follow the same procedures for all other servers.

After you do an offline defrag, you should make sure a backup is taken soon after. If for some reason you have to do a restore, and you have not done a backup since you did the offline defrag, the *ntds.dit* file on the domain controller will go right back to the size it was prior to the defrag.

Changing the DS Restore Mode Admin Password

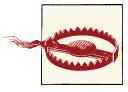
It is a good practice to periodically change the password for your domain administrator accounts. This should be done so that the password does not find its way to more people than it should, and so that you don't have former administrators trying to perform tasks they shouldn't if they are no longer in that AD group.

The domain administrator accounts should not be the only ones you are concerned about, though. The DS Restore Mode administrator account is just as important and

can be used to do very damaging things, such as directly modifying the contents of the Active Directory database. For this reason, you should also periodically rotate the DS Restore Mode administrator password.

You can use the *ntdsutil* command to change the DSRM password while the DC is online. The *reset password on server %s* subcommand can be used from the `set dsrm password` menu, where *%s* is the name of the server to target. Leave *%s* blank if you want to change the password on the local machine. In the following example, we set the password for the *DC1* domain controller:

```
ntdsutil: set dsrm password
Reset DSRM Administrator Password: reset password on server dc1
Please type password for DS Restore Mode Administrator Account: **********
Please confirm new password: **********
Password has been set successfully.
Reset DSRM Administrator Password: quit
```



You cannot use *ntdsutil* to set the DS Restore Mode administrator password if the target machine is currently in DS Restore Mode.

A feature introduced in Windows Server 2008 R2 enables you to copy the password of a domain account to the DSRM password on a domain controller using *ntdsutil*. This makes maintenance of the DSRM password across an entire domain much easier, since you can schedule a task on each DC to copy the password of the domain account on a regular basis.

This feature was also made available as a hotfix for Windows Server 2008 domain controllers. You can obtain this hotfix from MS Knowledge Base article 961320.

To take advantage of this feature, first create a disabled user account in AD in which you will maintain the DSRM password. Next, run the following *ntdsutil* commands. Be sure to substitute the username of the account you are using:

```
ntdsutil: set dsrm password
Reset DSRM Administrator Password: sync from domain account dsrm-sync-user
Password has been synchronized successfully.
Reset DSRM Administrator Password: quit
```

If you have multiple domains in your forest, you will need to create and maintain a user account to synchronize the DSRM password with in each domain.

One of the easiest ways to regularly rotate the DSRM password is to schedule a task using Group Policy Preferences. To do this, first create and link to the Domain Controllers OU a group policy called DSRM Password Sync Policy. Edit the group policy and navigate to *Computer Configuration\Preferences\Scheduled Tasks*. Create a new scheduled task, as shown in Figure 18-24. Use the following inputs to populate the task properties:

- Name: *DSRM Password Sync*
- Run: *%windir%\System32\ntdsutil.exe*
- Arguments: *"set dsrm password" "sync from domain account dsrm-sync-user" q q*
- Comments: *Synchronizes the DSRM Password with the dsrm-sync-user domain account.*

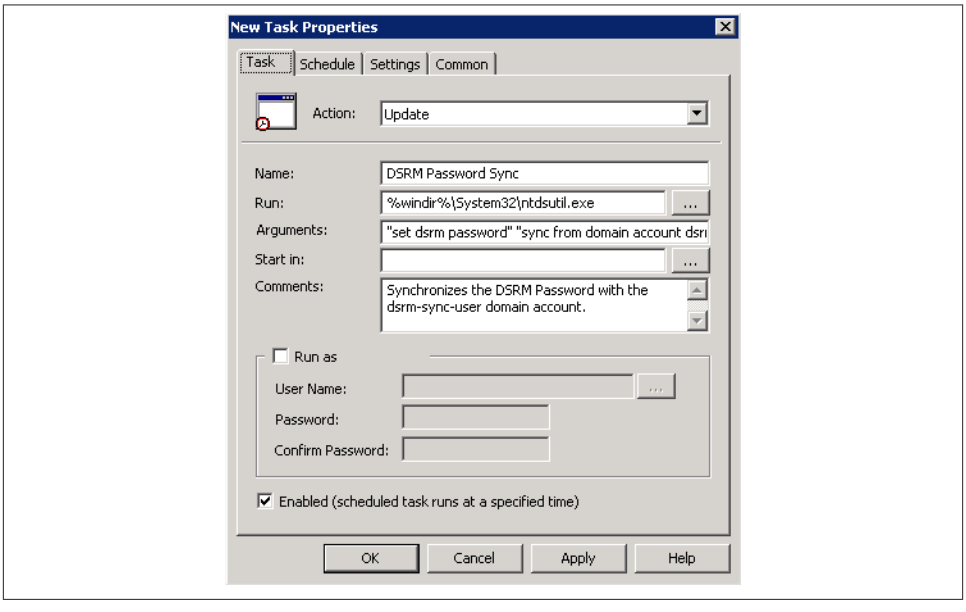


Figure 18-24. Creating the scheduled task

Schedule the task to run periodically (for example, once a day at 9 A.M.) on the Schedule tab. Once you have created the policy and allowed it time to apply, verify that it was created successfully by checking the DC's Task Scheduler as shown in Figure 18-25. Allow the scheduled time to pass and then reboot the DC into DS Restore Mode to ensure that the password was actually synchronized as expected.

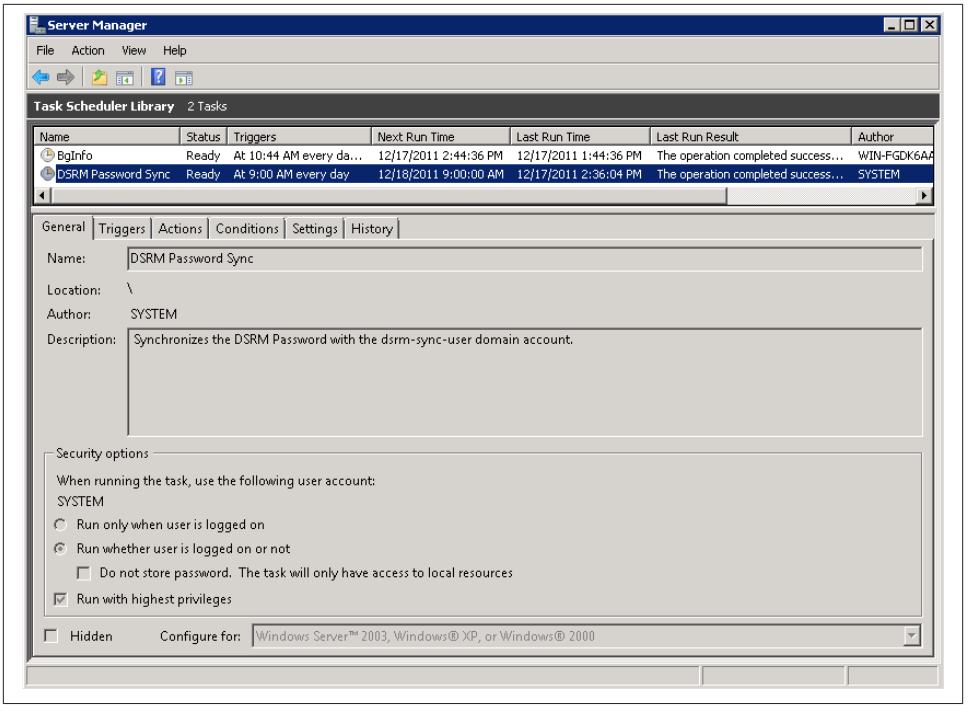


Figure 18-25. DSRM Password Sync scheduled task

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 restoring by replication, authoritative restores, nonauthoritative restores, and the Active Directory Recycle Bin. 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 19

Upgrading Active Directory

Like any other service or application, you must be prepared to upgrade Active Directory as it evolves with each version of Windows and as your organization's Information Technology group moves forward. Given the fundamental nature of Active Directory as a core IT service, there is undoubtedly trepidation when the discussion of an upgrade comes up.

With planning and due diligence, however, you can successfully upgrade your organization's Active Directory environment to take advantage of new features and continue to operate on a supported platform.

In this chapter, we'll take a look at the new features that have come with each version of Active Directory, some of the steps necessary to start planning your upgrade, and how to initiate the upgrade process. Finally, we'll look at some known issues that you should be aware of as you plan to upgrade Active Directory in your organization.

Active Directory Versions

Before we go too far, it's worth looking at the new features and functionality that have been added with each successive version of Active Directory. Active Directory is over a decade old and there continues to be innovation in each release to push the bar higher and deliver more value to Active Directory customers.

As you read this section, look at the features and functionality that come with each version of Windows ahead of you. You can use this information to start building a business case for the upgrade. Conversely, looking at the functionality available in newer versions of Windows may show you that a change won't really be of benefit to your organization and that upgrading isn't worth the cost and effort. If this is the case, wait for the next version of Windows and reevaluate.

One consideration you should keep in mind if you do decide to wait for the next version is the Microsoft product support lifecycle for the version of Windows you're running, and where your version will be when the next version of Windows comes out. The product support lifecycle defines how long Microsoft will take calls about a given product, produce security patches for the product, and fix bugs (and issue hotfixes). If you are out of support or will run past the support period for your version of Windows before the next version is released, you should plan an upgrade solely to make sure that you will be able to maintain support.

You can find Microsoft's support lifecycle policy online at *http://support.microsoft.com/lifecycle/*.

As you review the following sections, you'll find that we've grouped changes into new features and changes in functionality. We suggest that you review all of the new features applicable to your upgrade and then put them into the following categories:

- I would use the feature immediately.
- I would use the feature eventually.
- I would never use the feature or it is not important.

By "feature," we mean new functionality that is not just a modification of the way Active Directory worked in the previous version of Windows. In this sense, a feature is something you have to use or implement explicitly.

Rating each feature will help you determine how much you could benefit from the upgrade. As with the new features, we suggest you carefully review each of the differences and place them into the following categories:

- It would positively affect my environment to a large degree.
- It would positively affect my environment to a small degree.
- It would negatively affect my environment.

The vast majority of differences are actually improvements that should 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.

Windows Server 2003

Windows Server 2003 was the first upgrade to Active Directory since the first release in Windows 2000. Active Directory was an incredibly solid product in Windows 2000, but Windows Server 2003 brought new functionality, better performance, and enhanced security. We hope by now you are reading this section for background more than for planning an upgrade from Windows 2000!

New features

Application partitions
: You can create partitions (naming contexts) that can replicate to any domain controller in the forest. For more information on application partitions, refer to Chapter 4.

Concurrent LDAP binds
: Concurrent LDAP binds, also known as *fast binds*, do not generate a Kerberos ticket and security token and are therefore much faster than simple LDAP binds.

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 the two forest root domains. This requires the Windows Server 2003 forest functional level.

Domain controller rename
: The rename procedure for domain controllers requires a single reboot. Previously, renaming a domain controller was not possible without first demoting the DC and then renaming and re-promoting it.

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. This requires the Windows Server 2003 forest functional level.

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 linked to structural classes in the schema. With dynamic auxiliary classes, you can link a dynamic auxiliary class when creating an object (or add it after the creation). The auxiliary class does not need to be defined in the schema as an auxiliary class for the object's `objectClass`. This requires the Windows Server 2003 forest functional level.

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. Dynamic objects do not remain as tombstones when they are deleted due to expiration. This requires the Windows Server 2003 forest functional level.

Install from Media
: This much-needed feature allows replica domain controllers to be promoted into the domain using a backup from another domain controller. This can greatly decrease the amount of time it takes to promote domain controllers in large domains.

Last logon timestamp attribute
: A classic problem in a network environment is trying to determine the last time a user or computer logged on. The new `lastLogonTimeStamp` attribute is replicated, which means you can use a single query to find all users or computers that have not logged onto the network within a certain period of time. By default, this attribute is updated approximately every 10 days. This requires the domain to be operating at the Windows Server 2003 functional level. For more information on this attribute, visit *http://blog.joeware.net/2007/05/01/864/*.

MMC and CLI enhancements
: The Active Directory Users and Computers (ADUC) tool has been enhanced to allow multiselection of objects; other tools such as *repadmin* and *netdom* also have new options.

New DS CLI tools
: A new set of command-line tools provides greater flexibility for managing Active Directory from a command line. These tools include *dsadd*, *dsmod*, *dsrm*, *dsget*, and *dsquery*.

Group Policy RSoP
: Resultant Set of Policy (RSoP) has been built into ADUC and can be fully utilized with the Group Policy Management Console (GPMC). RSoP allows administrators to determine what settings of GPOs will be applied to end users and computers.

`objectClass` *change*
: You can change user objects to `inetOrgPerson` objects and `inetOrgPerson` objects to user objects. This requires the Windows Server 2003 forest functional level.

TLS support
: With Windows 2000, only the Secure Sockets Layer (SSL) was supported to encrypt traffic over the wire. Transport Layer Security (TLS), the latest standards-based approach for encrypting LDAP traffic, is now also supported.

Quotas
: In Windows 2000, if users had access to create objects, they could create as many as they wanted, and there was no way to limit it. Quotas allow you to define how

many objects a user or group of users can create. Quotas can also dictate how many objects of a certain `objectClass` can be created.

Redirect users and computers
: You can redirect the default locations to store new users and computers with the *redirusr* and *redircmp* commands, respectively.

Schema redefine
: You can mark attributes and classes as defunct and then redefine them in the schema in the event of a schema conflict, or if you simply want to hide unused classes and attributes. This requires the Windows Server 2003 forest functional level.

Universal group caching
: This feature eliminates the requirement to have a Global Catalog server present during login for universal group expansion. This feature is enabled at the site level and applies to any clients that log onto domain controllers in the site. Global catalogs are still needed for `userPrincipalName` authentications.

WMI filtering of GPOs
: In addition to the OU, site, domain, and security group criteria that can be used to filter GPOs, you can now use Windows Management Instrumentation (WMI) information on a client's machine to determine if a GPO should be applied. This functionality only works for Windows XP/2003 and newer clients.

WMI providers for trust and replication monitoring
: These new WMI providers provide the ability to query and monitor the health of trusts and replication programmatically.

Differences in functionality

Changes to the Pre-Windows 2000 Compatible Access group
: To enhance security, the Everyone security principal is no longer equivalent to all unauthenticated and authenticated users. Instead, it represents only authenticated users. To grant the equivalent of anonymous access in Windows Server 2003, the Anonymous Logon security principal must be added to the Pre-Windows 2000 Compatible Access group.

Distributed Link Tracking (DLT) service disabled by default
: The DLT service can be the source of thousands, if not millions, of `linkTrackOMTEntry` objects that are nestled within the System container of a domain. By default, the DLT service is disabled on Windows Server 2003 domain controllers.

Microsoft provides a script to clean up DLT objects in Knowledge Base article 315229.

Faster Global Catalog removal
: With Windows 2000, when you disabled the Global Catalog on a domain controller, the Global Catalog removal process could only remove 500 objects every 15 minutes. This has been changed so that the process is much quicker.

Intrasite replication frequency changed to 15 seconds
: The previous default intrasite replication interval of 5 minutes was changed to 15 seconds. This requires the Windows Server 2003 forest functional level for any Domain Controllers upgraded from Windows 2000.

ISTG and KCC scalability improvements
: The algorithms used to generate the intersite connections have been greatly improved, to the point where the previous soft limit of 300 to 400 sites has been raised to support roughly 3,000 to 5,000 sites. This new replication mechanism requires Windows Server 2003 Interim mode or the Windows Server 2003 forest functional level.

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 linked attributes, such as `member`, will only replicate the changes within the attribute and not the contents of the whole attribute whenever they are updated. This requires Windows Server 2003 Interim mode or the Windows Server 2003 forest functional level.

No Global Catalog sync for PAS addition
: With Windows Server 2003, whenever an attribute is added to the partial attribute set (PAS), a full Global Catalog resynchronization is no longer performed as it was with Windows 2000. This was especially painful to administrators of large, globally dispersed Windows 2000 domains.

Signed and sealed 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 digitally signed and also encrypted.

Single instance ACL storage
: Unique security descriptors are stored once no matter how many times they are used, as opposed to being stored separately for each instance. This feature can shrink your *ntds.dit* file by some 20%–40%.

Windows Server 2008

Windows Server 2008 introduced a number of compelling features, including support for running on Server Core, read-only domain controllers (RODCs), and fine-grained password policies.

New features

Administrative role separation
: Users who are not domain administrators can be securely delegated administrative control of RODCs without providing them access to the writable Active Directory (i.e., they cannot modify anything but the RODC).

ADMX repository
: Upgraded Group Policy template files can be stored once per domain in the Sysvol, thus greatly reducing the size of the Sysvol for many organizations.

Database snapshots
: Point-in-time snapshots of the Active Directory database can be taken as a basis for disaster recovery and other object restore and comparison operations.

DNS Server features
: Support has been added for new locator options, automatic configuration during install, background zone loading, and multicast DNS.

DFS-R Sysvol replication
: The Sysvol can now be replicated with the new DFS-R replication engine, which is much more reliable and scalable when compared to NTFRS.

Fine-grained password policies
: Password and account lockout policies can now be defined on a per-user basis. For more information on fine-grained password policies, refer to Chapter 12.

GlobalNames DNS zone
: This is a new type of DNS zone that can help pave the way to migrating away from WINS by resolving unqualified hostnames.

Group Policy Preferences
: Group Policy Preferences allows you to control numerous settings and Windows features that were previously only accessible via scripts.

Last-logon statistics
: Windows Vista and Windows Server 2008 clients can store detailed last-logon success and failure information directly on user objects in the directory. This feature was implemented for Common Criteria compliance.

Owner access rights
: An additional well-known security principal representing the owner of an object is now available.

Phonetic name indexing
: The `displayName` attribute is phonetically sortable on Japanese-locale domain controllers.

Read-only DNS
: RODCs can host dynamic DNS zones and refer any updates to writable domain controllers.

Read-only domain controllers
: RODCs do not allow local writes, and any compromise of these domain controllers will not replicate to writable domain controllers in the domain. They also do not store passwords and other secrets by default. This feature adds a great deal of security to domain controllers in locations with questionable physical security. For more information on RODCs, refer to Chapter 9.

Restartable directory service
: Active Directory can be stopped to allow for certain offline operations to be performed without restarting the domain controller in Directory Services Repair Mode.

Server Core support
: Domain controllers can now run on a version of the Windows Server 2008 operating system that is substantially lighter and thus more secure. Features such as the Explorer shell, Internet Explorer, and other items not essential to server operations have been removed from Server Core.

Starter Group policies
: Group Policy templates on which administrators can base new policies can be defined.

Differences in functionality

Auditing and logging infrastructure enhancements
: Auditing of Active Directory access and changes as well as various other actions has been updated substantially.

Delegated domain controller promotion
: Domain controller promotion can be broken up into a two-step process allowing delegation of the physical hands-on portion of promoting a domain controller to users other than domain administrators, such as local site support technicians.

Domain DFS scalability enhancements
: Domain-based DFS roots can host more than 5,000 links and can also leverage access based enumeration.

ESE single-bit error correction
: The JET database engine that Active Directory uses is now capable of detecting single-bit errors and correcting them, and thus reducing incidences of database failure due to corruption.

Kerberos AES key length upgrade
: The maximum key length supported by Kerberos for the Advanced Encryption Standard has been lengthened from 128 bits to 256 bits.

Windows Server 2008 R2

Windows Server 2008 R2 was a relatively small release for Active Directory, with regard to the number of new features. Most notably, the Active Directory Recycle Bin came with Windows Server 2008 R2.

New features

Active Directory Administrative Center
: The Active Directory Administrative Center (ADAC) is Microsoft's new GUI for managing Active Directory.

Active Directory Web Service
: This is an additional protocol endpoint that now runs on domain controllers and AD LDS servers. All of the Active Directory PowerShell cmdlets call the web service in lieu of making direct LDAP or RPC calls. Despite its name, there is no relation to traditional web server technologies such as Internet Information Services (IIS).

Authentication mechanism assurance
: With authentication mechanism assurance (AMA), administrators can define special SIDs that will be injected into a user's security token if that user logs in with a particular type of smart card certificate.

Kerberos forest search order
: Domain controllers can automatically search trusted forests for a Kerberos service principal name (SPN) when the SPN cannot be located in the domain controller's forest.

Managed service accounts
: Service accounts are an age-old security problem for Active Directory environments. Service account passwords are often rarely (if ever) changed and known by multiple people. Managed service accounts (MSAs) solve this problem by automatically rotating the account's password on a regular basis.

PowerShell cmdlets
: This was the first iteration of the Active Directory module for Windows PowerShell. The module included cmdlets primarily for managing users, groups, and computers.

Recycle Bin
: The Active Directory Recycle Bin (ADRB) enables administrators to recover deleted objects without resorting to backups and without losing any of the objects attributes. For more information on the ADRB, refer to Chapter 18.

Differences in functionality

Bridgehead server load balancing
: This improvement to the replication topology generation processes in Active Directory ensures that incoming replication connections are load balanced across servers in a site.

Offline domain join
: Windows 7 clients can be joined to the domain without network connectivity by pre-staging computer account information in Active Directory.

Weak encryption types disabled
: Windows Server 2008 R2 DCs disable support for DES and 3DES encryption types by default. This can be reverted by explicitly enabling these encryption types via Group Policy.

Windows Server 2012

Windows Server 2012 offers a substantial number of new features for Active Directory, as well as solutions for long-term challenges such as safely virtualizing domain controllers.

New features

Active Directory–based activation
: Windows 8 and Windows Server 2012 installations can now be activated based on licensing data published to Active Directory. This obviates the key management servers (KMSs) that were previously required for activation in enterprise environments.

Domain controller cloning
: Windows Server 2012 domain controllers that are running on a virtual machine generation ID (VM Gen ID)–aware hypervisor can be duplicated by cloning the domain controller's virtual hard disk files. For more information on domain controller cloning, refer to Chapter 9.

Dynamic Access Control
: Dynamic Access Control (DAC) functionality in Active Directory extends the security token to include claims. These claims are based on attributes of the user object in Active Directory as well as of the workstation the user logs in from. Compound expressions to evaluate claims can be applied to Windows Server 2012 file and folder permissions in lieu of complex group memberships. For more information about DAC, refer to Chapter 16.

Flexible authentication secure tunneling
: Domain controllers implement RFC 6113, also known as *Kerberos armoring*. This functionality enhances security by removing avenues for spoofing and dictionary attacks.

Group managed service accounts
: Managed service accounts (MSAs) have a significant limitation in that they cannot be used when the service must run across multiple servers. Group managed service accounts (gMSAs) solve this problem by enabling multiple servers to share an MSA and obtain the password when it changes.

Group Policy infrastructure status
: The Group Policy Management Console (GPMC) can now report on the health of the Group Policy information stored in Active Directory and on your domain controllers.

Differences in functionality

Active Directory Administrative Center enhancements
: The Active Directory Administrative Center GUI has been enhanced to include support for managing the Recycle Bin and fine-grained password policies. The UI is also substantially faster than in the Windows Server 2008 R2 release.

Cross-domain Kerberos constrained delegation
: Kerberos constrained delegation (KCD) can now be performed across domain boundaries.

Deferred index creation
: Indexing of Active Directory attributes can now be deferred until domain controllers are rebooted or indexing is explicitly requested. This resolves performance issues in large forests where indexing an attribute could cause a significant performance hit on all domain controllers as they created the index.

Deployment simplifications
: The classic dcpromo wizard and the *adprep* upgrade preparation utility have been retired in favor of an integrated deployment wizard in Server Manager that handles all facets of domain controller promotion and Active Directory upgrades. These steps can also be performed through new Windows PowerShell cmdlets.

Kerberos constrained delegation configuration changes
: KCD can now be configured on the service that is being delegated *to* rather than on the service performing delegation. These changes also remove the need for Domain Admin–level access in order to configure KCD.

Offline domain join DirectAccess support
: Offline domain join has been enhanced to also include the prerequisites for a machine to connect to DirectAccess. This enables machines to be joined to the domain over the Internet.

PowerShell enhancements
: The Active Directory module for Windows PowerShell has been enhanced to include cmdlets for managing replication and the domain controller lifecycle (promotion, demotion, etc.).

RID pool consumption warnings
: The size of the global RID pool has been doubled from approximately one billion RIDs to two billion RIDs. To prevent depletion of the RID pool without administrator knowledge, the system now logs events when 90 percent of the global RID pool is used. A soft block in issuing new RIDs is also put in place that the administrator must lift.

Virtualization safety
: Windows Server 2012 domain controllers that are running on a VM Gen ID–aware hypervisor are aware of changes that might cause a rollback of the domain controller's state. When these changes are detected, the domain controller implements safeguards to ensure that replication (and other functionality) is not impacted. For more information on this change, refer to Chapter 9.

Functional Levels

Now that you are sufficiently excited about the new features included in the different versions of Active Directory, we will cover how you can actually enable these features. *Functional levels* dictate what operating systems can run on domain controllers. By ensuring a minimum operating system version across a domain or the entire forest, you control what Active Directory features are available or can be enabled.

To illustrate why functional levels are necessary, let's look at two examples. First, consider the "last logon timestamp attribute" feature that was added in Windows Server 2003. With this feature, a new attribute called `lastLogonTimeStamp` is populated when a user or computer logs onto a domain, and it is replicated to all the domain controllers in the 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 domains, 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 consistently updated and truly useful, all domain controllers in the domain must 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 determine this by querying the functional level for the domain. Once they discover the domain is at a certain functional level, they can start utilizing features specific to that function level.

Likewise, there are times when all domain controllers in the forest must be running a certain operating system before a certain feature can be used. A good example is with the Active Directory Recycle Bin. Since the manner in which objects are deleted is changed when the Recycle Bin is enabled, and it is possible that an object (such as a group) might have a reference to an object in another domain, there must be a guarantee that every domain controller wo;; handle deletions in the same way. In order to make sure this is the case, every domain controller in the forest must be running at least Windows Server 2008 R2 for the Active Directory Recycle Bin to be available. Once all of the domain controllers are running Windows Server 2008 R2, the forest functional level can be upgraded and the Recycle Bin can be enabled.

Unlike previous versions of Windows, the Windows Server 2012 functional levels provide virtually no benefit. In Windows Sever 2012, Microsoft endeavored (quite successfully) to make all of the new Active Directory functionality available without the domain or forest functional level needing to be raised. Raising the domain functional level enables only two Kerberos features configured via Group Policy ("Always provide claims" [inside "KDC support for claims, compound authentication and Kerberos armoring"] and "Fail authentication requests when Kerberos armoring is not available"). Raising the forest functional level does not add any additional functionality.

Raising the Functional Level

You can raise the domain and forest functional levels with a couple of different tools. The Active Directory Domains and Trusts MMC snap-in is one option, regardless of the version of Windows you are running. If you're running Windows Server 2012, you can also use the Active Directory Administrative Center.

To raise the domain or forest functional level with ADAC, select the domain you want to modify in the navigation pane, and then click Raise Domain Functional Level or Raise Forest Functional Level in the task pane. The dialog will look similar to Figure 19-1. The functional levels you can move to will be listed in the drop-down. After you click OK, you will be notified that it may take some time for the change to replicate across the environment, as shown in Figure 19-2. Raising the functional level is replicated just

like any other change to the directory. For a listing of known issues that are associated with functional level changes, review this website.

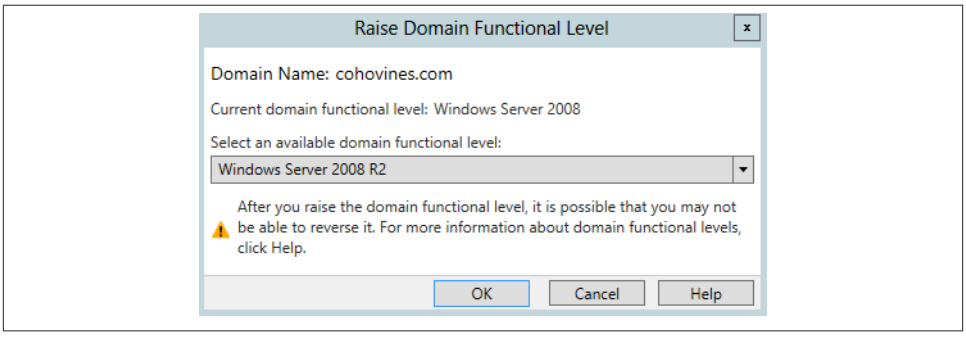


Figure 19-1. Raising the domain functional level in ADAC

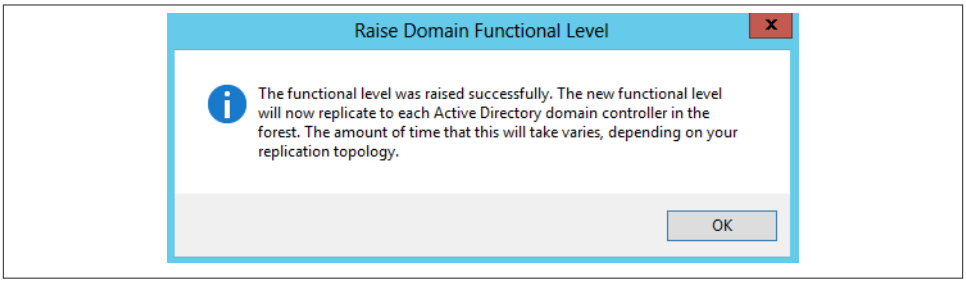


Figure 19-2. Domain functional level raise confirmation

To raise the functional level of a domain with Active Directory Domains and Trusts, open the snap-in, browse to the domain you want to raise, right-click on it in the lefthand pane, and select Raise Domain Functional Level. You will then see a screen similar to that in Figure 19-3.

Select the new functional level and click the Raise button. You will then get a confirmation that the operation was successful or an error stating why the functional level couldn't be raised. Figure 19-4 shows the message returned after successfully raising the functional level. Follow the same procedure to raise the functional level of a forest, but right-click on Active Directory Domains and Trusts in the lefthand pane and select Raise Forest Functional Level.

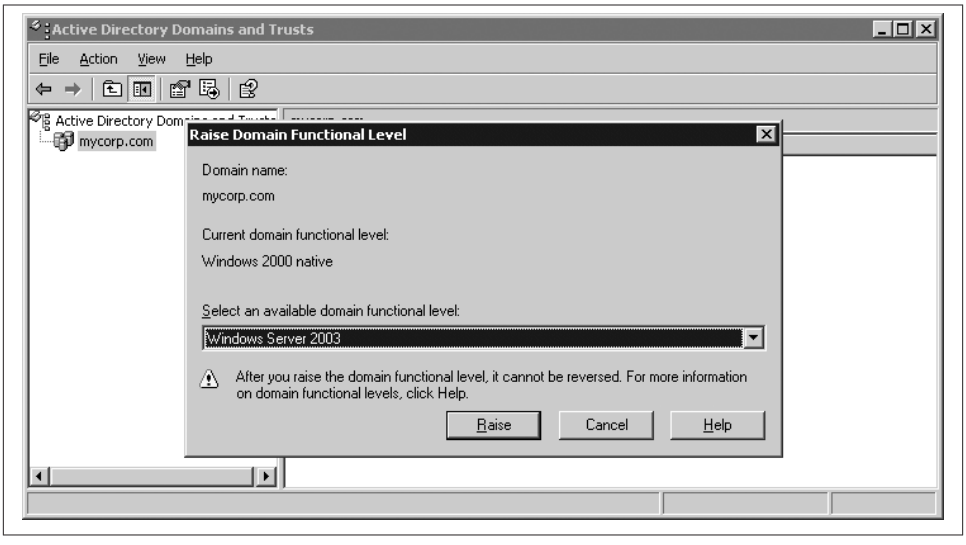


Figure 19-3. Raising the domain functional level with AD Domains and Trusts

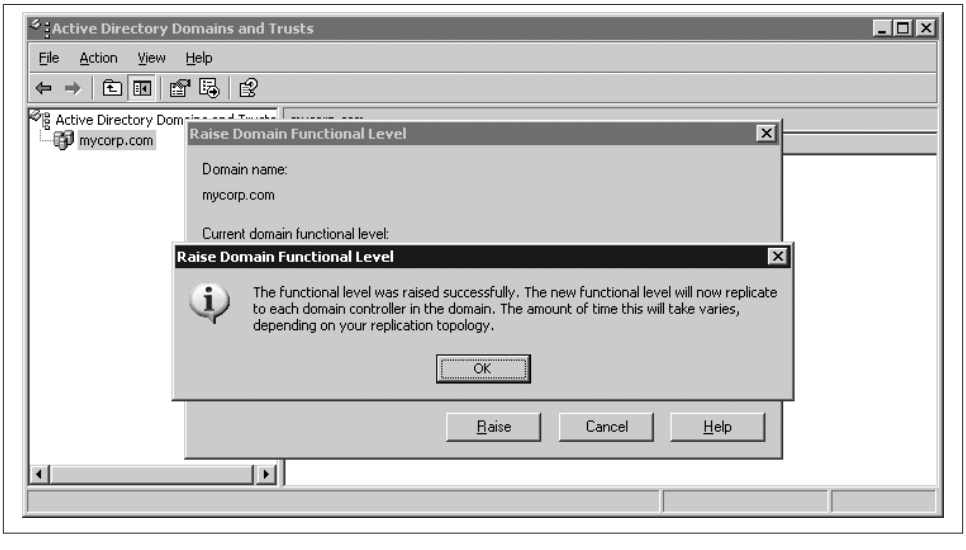


Figure 19-4. The result of raising the domain functional level with AD Domains and Trusts

You can determine the functional level of a domain or forest in a few other ways. First, you can look at the `msDS-Behavior-Version` attribute on the Domain naming context (e.g., `dc=cohovines, dc=com`) for domains or the Partitions container in the

Configuration naming context (e.g., `cn=partitions, cn=configuration, dc=cohovines, dc=com`) for the forest. Table 19-1 lists the values for this attribute and the functional levels they correspond with.

Table 19-1. Functional levels

msDS-Behavior-Version value	Functional level
0	Windows 2000
1	Windows Server 2003 Interim
2	Windows Server 2003
3	Windows Server 2008
4	Windows Server 2008 R2
5	Windows Server 2012

Windows 2000 did not have the concept of functional levels. Instead, you had a choice of Windows 2000 mixed mode, which supported Windows NT 4 domain controllers, and Windows 2000 native mode, which only supported Windows 2000 domain controllers. These two modes were not tracked in the `msDS-Behavior-Version` attribute.

Alternatively, you can view this information by simply looking at the *RootDSE* for a domain controller. The *RootDSE* contains two attributes that describe the current functional level:

domainFunctionality
: This value mirrors the `msDS-Behavior-Version` value on the Domain naming context.

forestFunctionality
: This value mirrors the `msDS-Behavior-Version` value on the Partitions container.

Functional Level Rollback

One of the challenges for administrators in organizations that implement change control procedures is that raising the functional level is not a step that can generally be reverted without a complete restore of a domain or forest. This makes obtaining approval for the change difficult, since the back-out plan section of the change request document is essentially blank.

Fortunately, starting with the Windows Server 2008 R2 functional level, you can roll back to the previous functional level if necessary. You can only revert to the Windows Server 2008 (or newer) functional level, though, and only if the Active Directory Recycle Bin has not been enabled.

To roll back the domain or forest functional level, you can use the *Set-ADDomainMode* and *Set-ADForestMode* PowerShell cmdlets. For example, if you want to roll back the *cohovines.com* domain to the Windows Server 2008 domain functional level, run this command:

```
Set-ADDomainMode cohovines.com -DomainMode Windows2008Domain
```

Beginning the Upgrade

When you are ready to introduce the first new domain controller into your environment, you'll need to make some updates. These updates include modifications to the Active Directory schema as well as various operational modifications, such as additional permissions for Active Directory to function correctly.

If you've upgraded Active Directory to any version after Windows 2000 before, you are undoubtedly familiar with the *adprep* utility on the Windows installation media. First you would run *adprep /forestprep* to upgrade the schema, and then you would move on to *adprep /domainprep* and *adprep /rodcprep*. Each of these steps had the potential to fail, and finding the utility was difficult in the first place.

Fortunately, when you introduce the first Windows Server 2012 domain controller, the promotion process will take care of everything for you now.

adprep is still available if you wish to perform the Windows Server 2012 preparation steps manually or in advance of introducing the first Windows Server 2012 domain controller.

One of the clever aspects of the *adprep* utility as well as the new Windows Server 2012 process is that it stores its progress in Active Directory. This is useful because it means it can gracefully recover from failures halfway through execution. It also provides a quick way to determine whether all of the necessary operations have completed and whether *adprep* was successful. Another benefit of storing the operations in Active Directory is that if you encounter problems and need to call Microsoft Product Support Services (PSS), you can look at this container and list out all of the operations that have been successful. PSS will then be able to look up which operation is failing.

A Forest Updates container is created directly under the Configuration container. Within the Forest Updates container are two other containers, one called Operations and the other called ActiveDirectoryUpdate. The Operations container contains additional containers, each one representing a certain task that *adprep* has completed. For example, one operation might be to create new `displaySpecifier` objects. The operation container names are GUIDs, and the objects themselves do not contain any information of interest.

The other container within the Forest Updates container is called ActiveDirectoryUpdate. This container is created and/or updated after the forest upgrade preparation process finishes. Figure 19-5 shows what these containers look like when viewed with the ADSI Edit snap-in. In addition to the ActiveDirectoryUpdate and Operations containers, there is also a Windows2003Update container created when you run the Windows 2003 version of *adprep*, and an ActiveDirectoryRodcUpdate container resulting from the RODC preparation step.

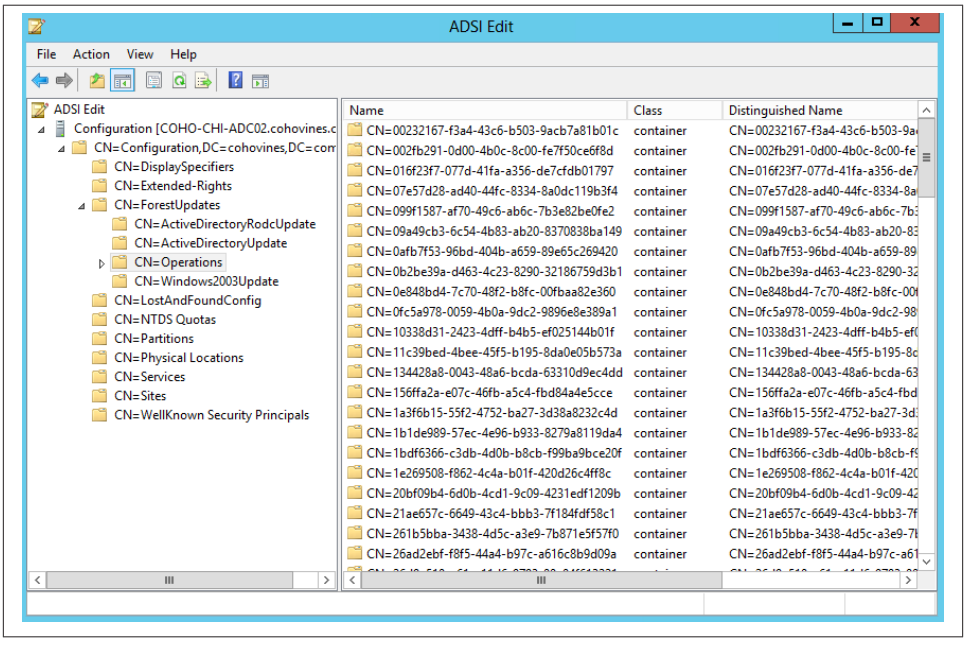


Figure 19-5. Forest Updates Operations container

Similar to the Forest Updates container, each domain has a Domain Updates container that tracks upgrades to the domain. You can find this container by browsing under the System container at the root of the domain, as shown in Figure 19-6.

Known Issues

While most Active Directory upgrades will go smoothly, there is always the potential for bumps in the road. Proper planning, testing, and communication in your environment are key to a successful upgrade project. We have been involved in countless Active Directory upgrade projects since Windows Server 2003 debuted, and each time we have seen a project fail it has been because one of these three elements (planning, testing, or communication) was left out.

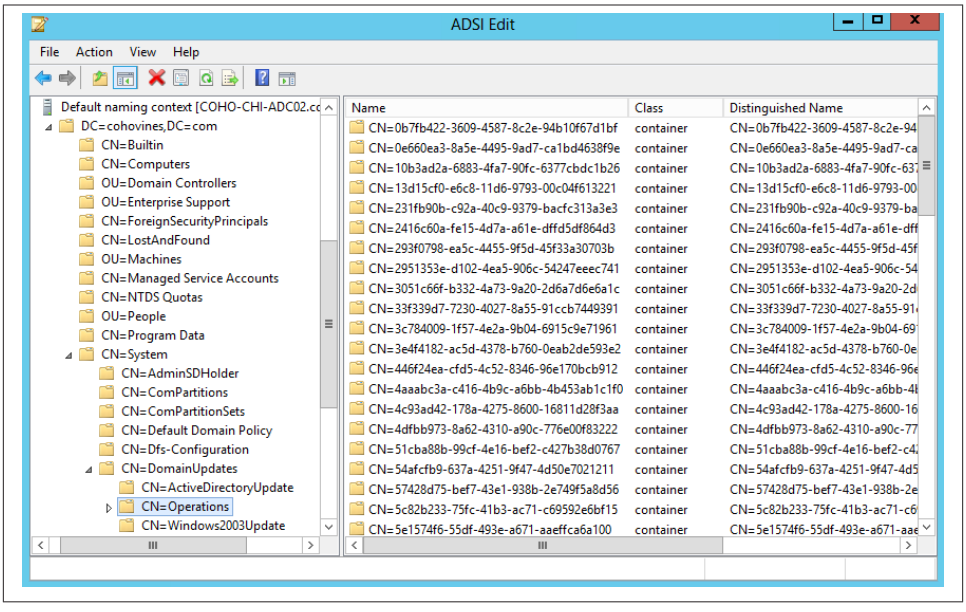


Figure 19-6. Domain Updates Operations container

Microsoft maintains an excellent document on TechNet that is updated frequently as Microsoft's support organization works through issues customers encounter while upgrading their forests. This document can be found at microsoft.com, and we highly recommend that you review it before embarking on your upgrade project.

Summary

In this chapter, we covered the new features and differences in functionality in each version of Active Directory since Windows Server 2003. We then discussed the functional levels as well as how to upgrade the domain or forest's functional level and roll back if necessary. Finally, we discussed the initial steps of preparing the domain and forest for new domain controllers.

CHAPTER 20

Active Directory Lightweight Directory Services

Shortly after Microsoft released Windows 2000 Active Directory, developers and administrators started asking for a standalone Microsoft LDAP service that was similar to Active Directory, but without the overhead (e.g., DNS and FRS requirements, Group Policy, and other domain pieces like Kerberos and legacy SAM interoperability)—basically, something light and easy to set up, play with, and tear back down as required. While you can do this with Active Directory, there tends to be additional cleanup and configuration required, and things unrelated to the LDAP functionality can get confused and cause it all to malfunction.

In November 2003, shortly after Windows Server 2003 Active Directory was released, Microsoft released Active Directory Application Mode (ADAM) V1.0 to the Web. This was the product that the developers and administrators had been asking for: Active Directory Lite. ADAM allowed developers and administrators to play with Active Directory on Windows XP or Windows Server 2003 and newer servers without promoting the localhost to a full domain controller. The only DNS requirement is resolution of the hostname. There is no FRS, no Kerberos, no Group Policy, and no extra domain stuff. In fact, ADAM runs nicely as a regular Windows application that can leverage any Windows domain authentication or local machine authentication that is available, as well as offering up its own authentication that is completely application-specific. It is just as happy in a domain as it is on a standalone machine.

After ADAM's release, Microsoft continued to invest in the product, and ADAM was bundled with Windows starting with Windows Server 2003 R2. With Windows Server 2008, ADAM was renamed to Active Directory Lightweight Directory Services (AD LDS) and made available as a server role that can be installed from Server Manager. This change has led to significantly enhanced visibility of AD LDS as a solution available to administrators.

As a result of this enhanced visibility, companies large and small (including Microsoft) have been building and deploying AD LDS–based applications. Due to the fact that AD LDS doesn't have any legacy requirements baggage other than being programmatically compatible with Active Directory, it is, by default, considerably more locked down out of the box. Where Active Directory followed the "on/enabled/open by default" mentality, AD LDS follows the "off/disabled/closed by default" approach. From a security standpoint, this is great. Another huge benefit is that because of their similarity, many experienced Active Directory administrators and developers will be able to get familiar with AD LDS quickly, facilitating administration, management, monitoring, and application development.

There are countless scenarios to which AD LDS can be adapted, and we could fill a small book discussing these scenarios. However, this chapter is simply intended to provide an introduction to the AD LDS toolset. Throughout this discussion, we will generally refer to Active Directory Lightweight Directory Services as AD LDS, but from time to time you may see references to ADAM, the previous acronym.

Common Uses for AD LDS

As you read this chapter, you will discover that AD LDS is extremely flexible. It introduces a number of new capabilities that enable use cases you might not have been able to accomplish with a traditional forest. To get you thinking about how AD LDS might be able to help your organization, let's take a minute to discuss a few common deployment scenarios.

The first scenario that we often see is the use of AD LDS as a test and development LDAP server for administrators and software developers alike. AD LDS can be installed on Windows 7 and used in the same manner as on a server. Administrators often use AD LDS to develop and test schema extensions in an isolated, disposable environment that mimics Active Directory. You can download AD LDS for Windows 7 from microsoft.com. Once you have downloaded the package and executed it, you can access Programs and Features from the Control Panel and click "Turn Windows features on or off" to install AD LDS and make the management tools discussed in this chapter available on your workstation.

You can also use AD LDS to provide a consolidated virtual directory that abstracts user objects across multiple trusted domains and forests into a single view for applications that expect all of the users to be in one location. Through the use of user proxies, discussed later in this chapter, you can create user objects for each user in your organization, but rely on your existing Active Directory domain(s) to store the users passwords. Since AD LDS can be installed multiple times on a single server (albeit with each instance listening on a different port), you could potentially run an instance of AD LDS for each application with customized LDAP directory requirements without requiring numerous servers or an expensive third-party virtual directory product.

Finally, some applications are beginning to take advantage of AD LDS as a lightweight, replicated, multimaster configuration store/directory. We're aware of a number of third parties who are doing this, in addition to various Microsoft products such as Exchange Server and the Forefront Threat Management Gateway.

AD LDS Terms

There are several terms used throughout this chapter. Here are some simple definitions to help you understand the concepts:

Instance
: An instance is a single installation of AD LDS on a server. Each server can have multiple instances of AD LDS installed; they will all be independently managed and use different LDAP and LDAPS ports.

Configuration set
: A collection of AD LDS instances that replicate with each other and share a common Schema and Configuration container. Application partitions can be shared among the instances that make up the configuration set as well.

Replica
: An instance of AD LDS that is part of a configuration set and replicates with other AD LDS instances.

Partition/naming context
: A unique namespace named after the root DN. AD LDS will have at least two partitions per instance: the configuration and schema partitions. AD LDS can also have multiple application partitions.

Application partition
: A type of partition that contains application data. AD LDS can have multiple application partitions per instance.

Configuration partition
: A type of partition that contains configuration data for the instance or configuration set.

Schema partition
: A type of partition that contains the class and attribute definitions.

Bindable object
: An object instantiated from an `objectClass` that has `msDS-BindableObject` listed as a static auxiliary class in the schema definition for the `objectClass`. The most common bindable object is the `user` class. Objects of this type are authenticated by AD LDS directly.

Bindable proxy object
: An object instantiated from an `objectClass` that has `msDS-BindProxy` listed as a static auxiliary class in the schema definition for the `objectClass`. The most common bindable proxy object is the `userProxy` class. Objects of this type are authenticated by AD LDS by proxying the authentication request to Windows.

Differences Between AD and AD LDS

AD LDS and AD are quite similar, but obviously there are differences, or else there would be nothing to talk about. This section isn't intended to be a comprehensive listing of all the differences, but rather an attempt to catch the major changes and popular gotchas.

Standalone Application Service

The most obvious difference is that AD LDS is set up to run as a standalone application service; it isn't a critical system-level service. This means that instead of the LDAP functions being handled by the *lsass.exe* process, they instead run from a *dsamain.exe* process. The not-so-obvious upshot of this is that AD LDS can be stopped or started on demand without having to reboot the machine. It also means AD LDS can be updated as needed (again, without rebooting the machine).

In addition to the benefit of stopping and starting on command, the new service implementation allows you to set up multiple instances of AD LDS on a single machine, each under a different service. So instead of having a single Active Directory instance on a machine responding to requests on port 389, you can have multiple AD LDS instances on a machine listening on various ports, with each instance having an entirely different schema.

Configurable LDAP Ports

Another difference is that with AD LDS you can actually control what ports the LDAP service is listening on. Active Directory seemed to have some registry entries you could change to specify the ports, but unfortunately, it was just a dirty trick: the ports were really hardcoded in the binaries so that you could modify them in the registry, but it didn't do anything.

No SRV Records

AD LDS doesn't register SRV records in DNS like Active Directory does; unfortunately, even registering your own SRV records on behalf of AD LDS doesn't make it so the DC location API calls will work for finding AD LDS instances. A network trace shows two reasons for this. The first reason is that the DC locator service doesn't look at the port field of the SRV record, so if AD LDS is running on a different port, the DC locator

service can't connect to it. The second reason is that the DC locator service sends a special UDP LDAP ping to the host, and AD LDS doesn't know how to respond to that request. This, of course, all makes sense since AD LDS isn't a domain controller, but it would be nice to have that functionality to make use of AD LDS a trifle more seamless for applications.

Instead of SRV records, AD LDS can publish `serviceConnectionPoint` (SCP) objects in Active Directory for resource location. These are objects in Active Directory that are usually published under the computer object on which the service is installed (this is configurable). The `serviceConnectionPoint` objects maintain key pieces of information about the AD LDS installation in the `keywords` and `serviceBindingInformation` attributes. An example of this information is listed in Example 20-1.

Example 20-1. AD LDS SCP example

```
>keywords: partition:DC=joeware,DC=net
>keywords: 1cf21445-f4f5-4451-bff2-d32f511c478b
>keywords: 42eca3c5-ee95-48c8-ab7d-7524ae887239
>keywords: partition:DC=domain,DC=com
>keywords: 9549ae64-a3ee-461a-918f-828f522b0382
>keywords: partition:CN=newpart
>keywords: 450d8cf9-19de-4c3c-817d-d5443fb968f5
>keywords: partition:CN=user
>keywords: partition:DC=test,DC=etherpunk,DC=local
>keywords: 7547aa81-51ad-416f-9ee6-7bfd1f88853d
>keywords: partition:DC=mytest,DC=com
>keywords: c4d69674-b673-45f9-ad14-b1e488d4a0c8
>keywords: 0a891bbf-1be6-4fe4-9406-3f21355e144e
>keywords: partition:DC=set-con,DC=org
>keywords: 81b58144-7068-410b-bd93-8248109a329b
>keywords: partition:DC=etherpunk,DC=local
>keywords: partition:CN=Configuration,CN={E28AE3C2-1228-4F6B-917C-56B9757DB796}
>keywords: 47abf207-536a-4ea4-9295-9ef3c7f1fb8c
>keywords: fsmo:naming
>keywords: fsmo:schema
>keywords: instance:ADAM1
>keywords: site:Default-First-Site-Name
>keywords: 1.2.840.113556.1.4.1791
>keywords: 1.2.840.113556.1.4.1851
>keywords: 14b5dcce-8584-4100-8d5b-139b1e8a95b1
>serviceBindingInformation: ldaps://adamserver.mycorp.com:636
>serviceBindingInformation: ldap://adamserver.mycorp.com:389
```

As you can see in the example, the `serviceBindingInformation` attribute shows the actual hostname and ports in use:

```
>serviceBindingInformation: ldaps://adamserver.mycorp.com:636
>serviceBindingInformation: ldap://adamserver.mycorp.com:389
```

In the `keywords` attribute, there is more good info. Here are some of the more useful pieces of information about the partitions that are available:

```
>keywords: partition:DC=joeware,DC=net
>keywords: partition:DC=domain,DC=com
>keywords: partition:CN=newpart
>keywords: partition:CN=user
>keywords: partition:DC=test,DC=etherpunk,DC=local
>keywords: partition:DC=mytest,DC=com
>keywords: partition:DC=set-con,DC=org
>keywords: partition:DC=etherpunk,DC=local
>keywords: partition:CN=Configuration,CN={E28AE3C2-1228-4F6B-917C-56B9757DB796}
```

And here are the FSMOs held by this instance:

```
>keywords: fsmo:naming
>keywords: fsmo:schema
```

the site the instance is in:

```
>keywords: site:Default-First-Site-Name
```

the instance name that maps to the actual service name on the host:

```
>keywords: instance:ADAM1
```

and some OIDs describing functionality of the instance:

```
>keywords: 1.2.840.113556.1.4.1791
>keywords: 1.2.840.113556.1.4.1851
```

If you choose, you can add additional keywords to help your applications select the proper instance or naming context. For more on using the `serviceConnectionPoint` objects for AD LDS, see the following:

- Administering AD LDS Service Publication
- Service Connection Points (SCPs) and ADAM/AD LDS
- Service Connection Points for Replicated, Host-based, and Database Services

No Global Catalog

Active Directory has the global catalog to allow you to easily query for objects across the entire forest hierarchy. You send a standard LDAP query to a special port, and you have access to a subset of attributes for all objects in the forest. A single query will allow you to search across the Configuration container, the schema, and every default domain partition in the forest. AD LDS has multiple partitions, but it doesn't have the matching global catalog functionality. Fortunately, the Windows LDAP server accepts a special LDAP control setting that allows you to use one single query that will scan across all of the partitions (or a subset of the partitions, like `DC=com`) of an AD LDS instance. Unfortunately, this functionality is available only to LDAP API–based applications, not ADSI scripts and programs.

We just described the `LDAP_SERVER_SEARCH_OPTIONS_OID` LDAP server control. This control, combined with the `SERVER_SEARCH_FLAG_PHANTOM_ROOT` flag, enables the special full instance search functionality. The control is also available for Active Directory, but generally isn't required due to the global catalog functionality. For more information, see this link.

There is an option that could work for ADSI and any third-party applications that don't implement the proper LDAP control. It involves implementing GC functionality yourself by setting up another top-level application partition and synchronizing certain objects and attributes from the other application partitions. You then grant read-only access to applications and users that need to query the entire instance, and full control to the syncing application. If you are simply after a quick lookup of a DN for an object based on a couple of public attributes like `sAMAccountName`, this option might work for you.

The loss of the global catalog also impacts a special protocol used by Outlook and Exchange, primarily for Address Book functionality. The Name Service Provider Interface (NSPI) and the backend Address Book support functionality are only provided in the Global Catalog code. Removing the GC functions also removes the NSPI and Address Book functions.

Top-Level Application Partition Object Classes

In Active Directory, application partitions must all be objects of the class `domainDNS`. AD LDS allows application partitions to be of any class, as long as it is a container-type object. This means that you can have application partitions that are of many different types, including `organizationalUnit`, `container`, `o`, `c`, `l`, or even `user`.

Group and User Scope

The scope of an AD LDS group or user is limited to the partition in which it exists; e.g., a group or user created in partition 1 can neither be used to assign permissions on objects in partition 2 nor added to a group in partition 2. You can, however, add Windows security principals and AD LDS security principals from the configuration partition to the groups in other partitions.

FSMOs

In an Active Directory forest, you have two FSMOs for the entire forest (the schema and domain naming masters) and three FSMOs (the PDC emulator, RID master, and infrastructure master) for each domain. A single-domain forest would have a total of five FSMOs, a two-domain forest would have eight FSMOs, and so on. AD LDS, on the

other hand, only has two FSMO roles regardless of the number of partitions or the number of replicas. The two FSMOs are the naming master and the schema master, whose roles map to the Active Directory domain naming master and schema master. Just like with Active Directory, if you want to add a new partition, you must contact the naming master, and if you want to update the schema, you must contact the schema master.

The missing FSMOs are generally not too surprising when you think of the functional changes between AD and AD LDS. However, the lack of the RID master role may confuse some administrators because AD LDS still uses security identifiers (SIDs), and SIDs need to be unique to be of value. The RID master in an Active Directory domain ensures that each security principal that is created has a unique SID value to uniquely identify it within the domain. This is handled in a well-known way in Active Directory: it follows the basic mechanism used for NT4, where RIDs are incremented as they are passed out. AD LDS has changed this process so that a RID master is no longer needed.

Instead of using RIDs, AD LDS has an algorithm in place based on the GUID generation algorithm to generate unique SIDs. Instead of the SIDs having just a unique RID, the last four subauthorities in the SID are all generated for each user. While one or more of those subauthorities may match those of other groups or users, there shouldn't be a collision with all four values because AD LDS tries to enforce the uniqueness. This means a RID master is not needed. The sidebar "Windows SIDs Versus AD LDS SIDs" describes the differences in AD LDS security identifiers. For more information on Windows SIDs, see the sidebar in Chapter 2, "What's in a Security Identifier (SID)?" (page 17).

Windows SIDs Versus AD LDS SIDs

As you'll recall from Chapter 2, a Windows SID is generally composed of 2 fixed fields and up to 15 additional fields, all separated by dashes like so:

```
S-v-id-s1-s2-s3-s4-s5-s6-s7-s8-s9-s10-s11-s12-s13-s14-s15
```

The first fixed field (`v`) describes the version of the SID structure. Microsoft has never changed this, so it is always 1.

The second fixed field (`id`) is called the identifier authority. In Windows domains and Windows computers it uniquely identifies the authority involved, such as NULL (0), World (1), Local (2), NT Authority (5), etc.

The next 15 fields (`s1–s15`) are not required for every SID, and in fact, most SIDs only have a few of these fields populated. These additional fields are called subauthorities and help uniquely identify the object being referenced. The last subauthority on most SIDs is generally called the RID. This is the value that a domain or computer increments to create unique SIDs.

AD LDS changed the use of the SID fields for AD LDS Trustees and Security Principals slightly. The new format looks like this:

```
S-v-id1-id2-r1-r2-r3-r4
```

The first fixed field (`v`) still describes the version of the SID structure. Although the use of the various fields has been slightly modified, the binary format of the SID structure is identical; Microsoft didn't change the version, and it is still always 1.

The second and third fixed fields (`id1-id2`) are the identifier authority and are specific to the directory partition. Each partition in an AD LDS instance will have a unique combination of these values.

The next four fields (`r1-r4`) are not required for every SID. The built-in groups will only have `r1` populated, and they will be populated with well-known RID values. The built-in groups are Readers (`514`), Users (`513`), and Administrators (`512`). Any other objects created that need SIDs will have `r1-r4` populated with unique combinations of randomly generated values.

It is possible to encounter Windows SIDs inside of AD LDS. They are easily identified because they follow the Windows SID format instead of the AD LDS format. These Windows SIDs are used to either identify Windows Trustees or grant permissions inside of AD LDS to Windows Trustees. For example, you will add the SID for a Windows user to the Administrators group of any partition in order to grant that Windows user administrator rights over the partition.

Schema

The default AD LDS schema is very lightweight in comparison to the default AD schema. Initially, it only has 44 classes and 268 attributes, which is quite sparse. Microsoft supplies a couple of LDIF files with AD LDS so you can build up the schema to instantiate some Microsoft standard objects, such as `inetOrgPerson` and `user` objects. Extension of the schema beyond this is fully supported and similar to the process used to extend the AD schema.

Another difference is that you can have multiple schemas on a single machine with AD LDS. Each instance of AD LDS can have its own schema, and you can install multiple instances.

Service Account

AD runs in the LSASS process on domain controllers as Local System. The AD LDS service runs as *NT Authority\Network Service*, or you can specify any normal Windows user ID.

Configuration/Schema Partition Names

AD LDS, like AD, has both configuration and schema partitions. Unlike with AD, however, there is no root domain or root partition, so the names of these critical partitions are based on a randomly generated GUID and have nothing to do with the application partitions that exist in the instance. For example:

```
CN=Configuration,CN={E28AE3C2-1228-4F6B-917C-56B9757DB796}
```

Default Directory Security

The default security in Active Directory is extremely open. By default, any user that is from the domain or any trusted domain can look at most attributes of most every object. Attempts to lock down AD can be met with frustration, confusion, and applications not working correctly. AD LDS completely reverses this. By default, normal users do not have read access to anything but the schema. The only access control entries (ACEs) defined on the application partitions are read permissions for the Readers group, full control for Administrators, and minimal permissions for replication for the Instances group used for replicas.

User Principal Names

Active Directory user principal names follow RFC 822 email address formatting rules. AD LDS has no such limitation; instead of using *joe@mycorp.com*, you can simply use *joe* for the UPN. While Active Directory doesn't enforce `userPrincipalName` value uniqueness, AD LDS does in fact enforce this uniqueness.

Authentication

There are a few differences that must be mentioned in the area of authentication. To start with, there is some good news: you aren't locked into the types of objects that you can bind with. Any object can be used for binding, as long as it is derived from a class that includes the `msDS-BindableObject` auxiliary class in the schema definition and has a valid password value set for `unicodePwd`.

The next difference is that AD LDS bindable objects can only use simple bind or digest authentication. This means that if you want secure binding with AD LDS users or other AD LDS bindable objects without digest authentication, you must set up a certificate and encrypt the communication channel using SSL. In order to use secure authentication without SSL, you have to use Simple Authentication and Security Layer (SASL) binds with either Windows local or domain users. By default, an AD LDS user can be authenticated over the standard LDAP port with a clear-text simple bind. You can override this and force an SSL requirement by setting `RequireSecureSimpleBind=1` in the `msDS-Other-Settings` attribute of the object `CN=Directory Service,CN=Windows NT,CN=Services,CN=`<*Instance GUID*> in the configuration partition.

The last difference to mention here is bind redirection. AD LDS allows you to configure a bindable proxy object that links to a Windows user. The user can use a simple bind to AD LDS, which then proxies the authentication request to Windows in a secure manner. This is used when an application cannot use the Windows security principals directly, or possibly if you want to disguise the backend Windows account. Bindable proxies will also proxy password changes back to AD.

Unlike with AD LDS user simple-bind authentication, the default for using a simple bind for `userProxy` objects is to require an SSL connection. This can be overridden by setting `RequireSecureProxyBind=0` in the `msDS-Other-Settings` attribute of the CN=`Directory Service` object described earlier.

Users in the Configuration Partition

AD LDS allows you to create `user` and `userProxy` objects in the configuration partition. These users can then be added to any group in any partition. So, you could add the user to the configuration partition's Administrators group, which is nested in the Administrators group of all application partitions. This change allows an AD LDS user to administrate an AD LDS instance instead of having the requirement to use a Windows user. To enable this new capability, set `ADAMAllowADAMSecurityPrincipalsInConfig Partition=1` in the `msDS-Other-Settings` attribute of the CN=`Directory Service` object described earlier.

New and Updated Tools

One of the biggest initial complaints (and deployment blockers) with early versions of AD LDS was a lack of tools. Microsoft helped address this problem by creating new tools like *ADSchemaAnalyzer* and *ADAMSync* as well as updating existing AD tools like AD Sites and Services and LDP to work with AD LDS. We'll look at some of the available tools shortly.

AD LDS Installation

The AD LDS installation is broken up into two main pieces: installation of the base components and installation of new instances. The base component installation only needs to occur once per server, while new instance installations will occur every time you want to install a new instance or a replica of an existing instance.

Installing the Server Role

The base installation is as simple as launching Server Manager and adding the Active Directory Lightweight Directory Services role. Once the installation completes, you can use the AD LDS setup wizard to configure one or more instances on the server.

Installing a New AD LDS Instance

Once you have completed the server role installation, you will probably want to install a new instance of AD LDS. This process is also quite simple, though not as trivial as the server role installation. When you install a new instance, you need to have answers to the questions listed in Table 20-1.

Table 20-1. New instance installation questions

Question	Description	Example
Unique instance or replica? (Figure 20-2)	Do you want to replicate the configuration and schema with an existing instance, or do you want a completely new instance?	Unique instance
Instance name? (Figure 20-3)	This value is appended to the string "ADAM_" to specify the AD LDS service name running this instance and listed in the Add/Remove Programs dialog.	AddressBook
LDAP port number? (Figure 20-4)	The TCP port you should listen on for standard LDAP requests.	389
SSL port number? (Figure 20-4)	The TCP port you should listen on for SSL-based Secure LDAP (LDAPS) requests.	636
Create application directory partition? (Figure 20-5)	Select whether you would like to have the instance creation process create an application partition. This is recommended when you first set up AD LDS.	Yes
Partition name? (Figure 20-5)	The name to use for the initial application partition, only needed if you decide to have AD LDS create the initial partition during instance installation.	`CN=addressbook`
Data file location? (Figure 20-6)	Where do you want the AD LDS data files located?	*E:\AD LDS\AddressBook \data*
Data recovery files? (Figure 20-6)	Where do you want the AD LDS log files located?	*E:\AD LDS\AddressBook \data*
Service account? (Figure 20-7)	Service account to use for running this AD LDS instance. Network Service is recommended unless the server is a member of an NT4 domain or is in a workgroup and you need replicas.	Network Service
Initial administrator? (Figure 20-8)	What group or user should be initially assigned administrative rights over this instance?	Currently logged on user
Import LDIF files? (Figure 20-9)	Specifies whether or not you would like to import some of the basic MS schema definitions for `user`, `user Proxy`, `inetOrgPerson`, and Authorization Manager.	Import
Files to import? (Figure 20-9)	Select which of the displayed LDIF files you would like to import.	*MS-InetOrgPerson.LDF* *MS-User.LDF* *MS-UserProxy.LDF*

When you are prepared with responses to all of the questions in Table 20-1, you initiate the instance installation by selecting the Active Directory Lightweight Directory Services Setup Wizard from the Administrative Tools section of the Control Panel or the Start menu (Figure 20-1).

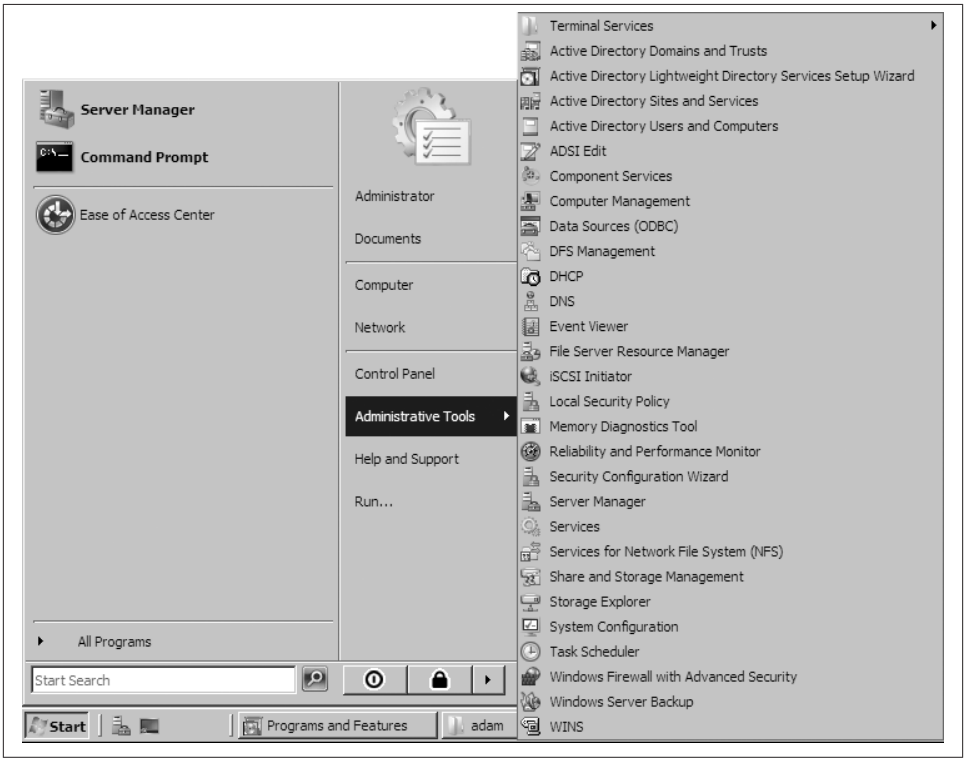


Figure 20-1. Launching the AD LDS Setup Wizard

You can also run *%windir%\ADAM\ADAMinstall.exe* to launch this wizard.

This will launch a normal Windows installation wizard. The first dialog of consequence (see Figure 20-2) asks whether you want to install a unique instance of AD LDS or create a replica of an existing instance. Replicas will be covered in the section "Installing an AD LDS Replica" on page 585; for now, choose "A unique instance" (the default). When you install a unique instance, you are creating an instance that has no connection to any other AD LDS instance and that has a fresh AD LDS schema and all the default configuration settings. You will not be able to configure this instance to replicate with any

other instances that currently exist, though you could create another new instance and install it as a replica of this instance.

Figure 20-2. Setup Options dialog

The next dialog (see Figure 20-3) allows you to specify the name of the instance. This name is appended to the string `ADAM_` and used for the name of the service running the instance. The name is also used in the Add/Remove Programs dialog, so you can later uninstall this instance if necessary. The default instance name for the first instance installed on a server is *Instance1*. If you use the name *Instance1*, the next time the default instance name will be *Instance2*, and so on.

After you choose your instance name, you will be presented with a dialog (see Figure 20-4) that allows you to specify the ports to use for standard LDAP and LDAPS. The default ports are 389 and 636; you should stick with these ports unless you know that your applications are flexible enough to allow you to specify different ports.

The only partitions created by default in AD LDS are the configuration and schema partitions. The instance installation allows you to create one initial application partition in the Application Directory Partition dialog (see Figure 20-5). The default for this dialog is "No, do not create an application directory partition." If you change the default, enter the DN of the partition you would like to create in the "Partition name" field.

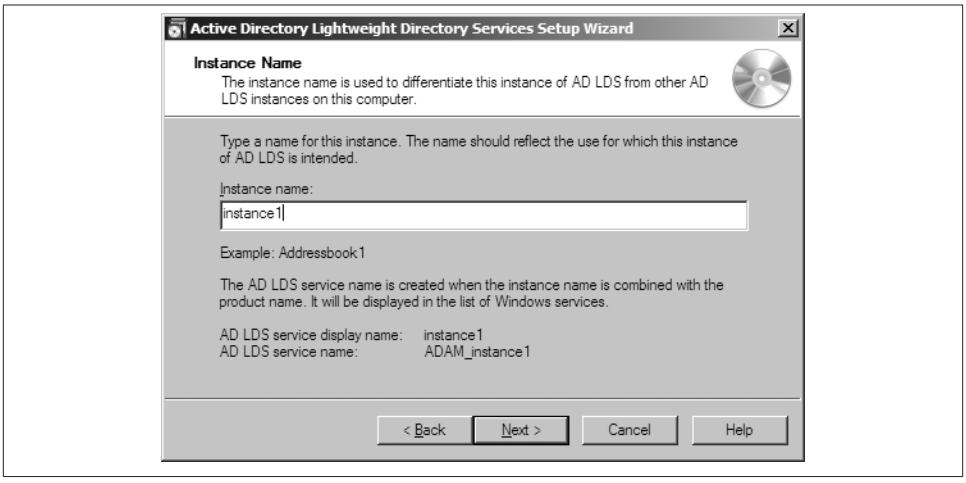


Figure 20-3. Instance Name dialog

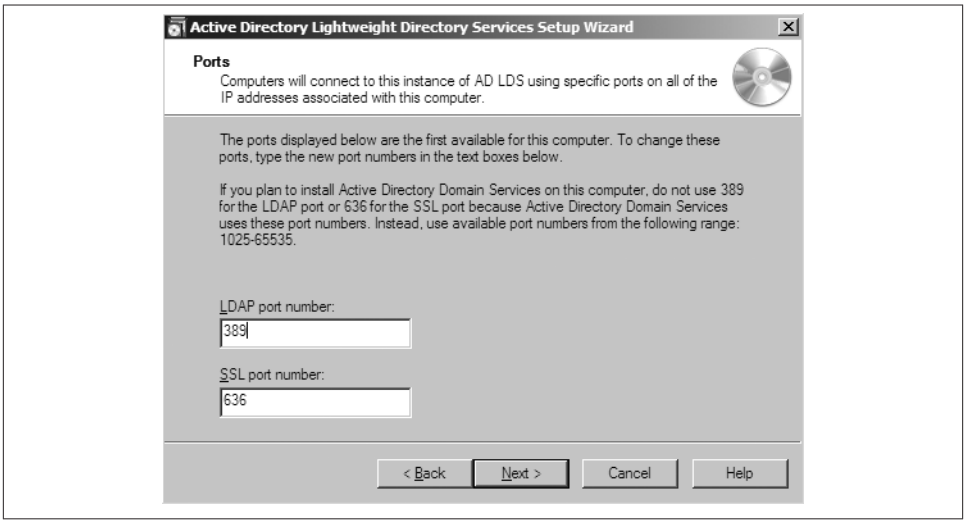


Figure 20-4. LDAP/LDAPS port selection

Figure 20-5. Application Directory Partition dialog

AD LDS allows you to choose where to place your database and log files (see Figure 20-6). The database file location is specified in the "Data files" field and includes the DIT file, as well as the temporary database EDB and CHK files. The transaction log file location is specified in the "Data recovery files" field and includes the log files that hold the database transaction information. In high-performance configurations, depending on the disk configuration of the server, it is usually recommended to split up these two sets of files onto separate physical drives. At the very least, you may want to move them from the default location, which is on the system drive in the *Program Files\Microsoft ADAM\<instancename>* folder.

Microsoft has written AD LDS so that, by default, it runs as the *Network Service* account. This is a very good thing, and you should generally stick with this default. However, if you do not want to run as *Network Service*, you have the option to change the security context to any Windows user you choose (see Figure 20-7). You should only need to make this change if you are running in a Windows NT4 domain, are running in workgroup mode and want to have replicas, or need to provide access to a load-balanced set of AD LDS servers with a shared name. In these cases, AD LDS will need to use a dedicated Windows account.

The next dialog you encounter is critically important; it is the dialog where you specify the initial administrator(s) of the AD LDS instance (see Figure 20-8). The default is the user installing the instance, but consider using the local Administrators group instead or, better yet, creating a domain group specific to administering this AD LDS instance.

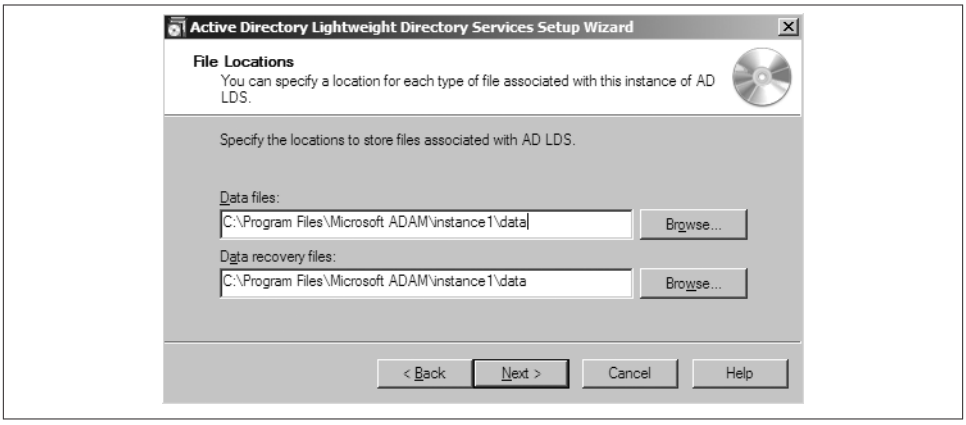


Figure 20-6. File Locations dialog

Figure 20-7. Service Account Selection dialog

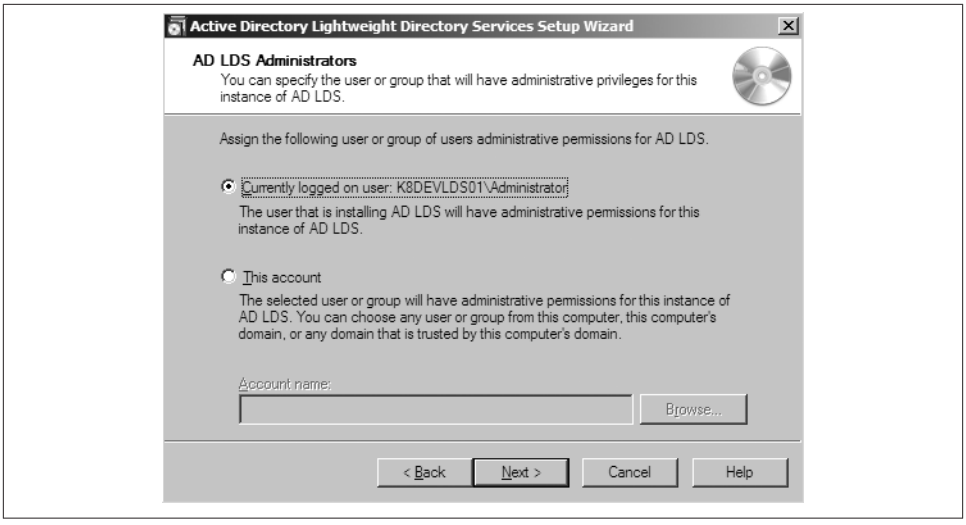


Figure 20-8. AD LDS Administrators dialog

Quite unlike with most other Windows applications, the server's Administrators group has no control over AD LDS's internal data or configuration unless the group is specifically added to the Configuration container's Administrators group (or role, if you prefer). This is great from a security standpoint, unless you somehow lose access to the users placed in that group; if that happens, you have lost control of the instance.

The final dialog that requires a decision, shown in Figure 20-9, concerns what LDIF files AD LDS should import. The default is that no files should be imported, which gives you the AD LDS base schema only. However, if you want to add the functionality gained through these LDIF files, you should allow the installation to import them. If you decide at a later time that you would like to incorporate any of these schema updates, simply use *ldifde.exe* to import them. The LDIF files are located in the folder *%windir%\ADAM*.

Table 20-2 shows a description of these LDIF files.

Table 20-2. AD LDS instance installation LDIF files

Filename	Contains
MS-AdamSyncMetadata.LDF	Metadata schema extensions that are required to utilize AD LDS Sync.
MS-ADLDS-DisplaySpecifiers.LDF	Display specifiers schema and data required to use the GUI tools with AD LDS.
MS-AZMan.LDF	Objects and attributes needed for Microsoft Authorization Manager (AZMan). AZMan is a framework for roles-based access control.
MS-InetOrgPerson.LDF	`inetOrgPerson` object definition.
MS-User.LDF	Active Directory `user` definition. Bindable object class, trustee exists in AD LDS.
MS-UserProxy.LDF	AD LDS `userProxy` definition. Bindable object class, trustee exists in Windows (either local or domain).

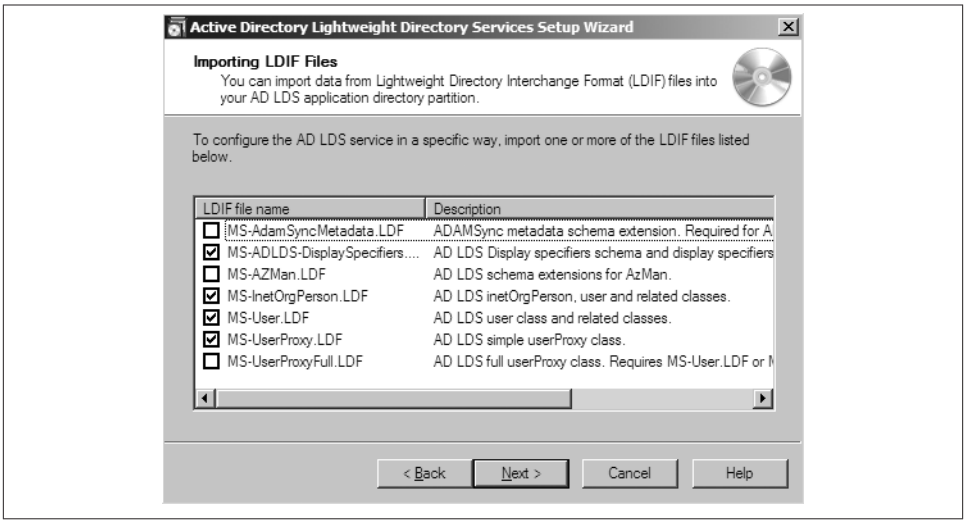


Figure 20-9. AD LDS LDIF import dialog

Filename	Contains
MS-UserProxyFull.LDF	AD LDS `userProxy` definition with additional attributes from the `user` class. Bindable object class, trustee exists in Windows (either local or domain).

After completing the LDIF file dialog, you will be presented with a "Ready to Install" dialog listing the choices you've made. Click Next if you accept the choices; the AD LDS instance will be created, and you will be rewarded with a dialog indicating that you successfully completed the AD LDS Setup Wizard. Click on Finish to complete the installation process.

Installing an AD LDS Replica

Installing an AD LDS replica, also known as joining a configuration set, is very similar to creating a new, unique AD LDS instance. In addition to the unique instance installation questions, you must answer several questions related to which configuration set you would like to join and what to replicate.

When you join a configuration set you are choosing to share the configuration and schema partitions with another AD LDS instance (or replica), just like domain controllers in the same forest. In addition, you have the option to replicate all, some, one, or none of the application partitions with all, some, one, or none of the other replicas in the configuration set. Figure 20-10 shows a complex configuration set shared between the servers *SERVER1*, *SERVER2*, and *SERVER3*.

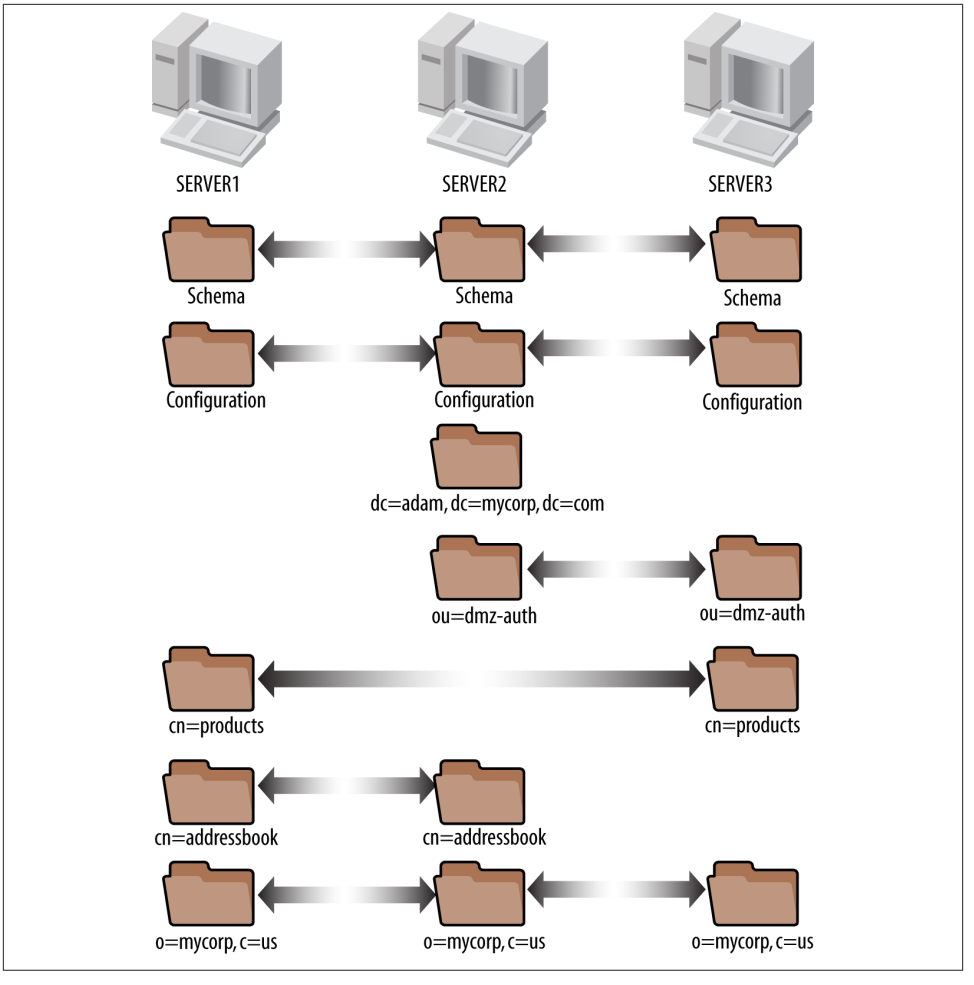


Figure 20-10. Complex configuration set

This may seem very complicated, but it is actually quite simple to configure. You need a plan of what partitions should go where; then you simply install each replica one by one, setting up the proper application partitions for replication.

The additional questions you need to respond to when installing a replica into a configuration set are listed in Table 20-3.

Table 20-3. Replica installation questions

Question	Description	Example
Source server? (Figure 20-12)	FQDN of any server in the configuration set to replicate initial configurations from.	*k8devlds01*
Source server LDAP port? (Figure 20-12)	LDAP port of instance to join.	389
Configuration set administrative credentials? (Figure 20-13)	Credentials of user with administrative permissions in configuration set.	MYCORP\administrator
Application partitions to replicate? (Figure 20-14)	Which application partitions, if any, you want to replicate to this specific replica.	`cn=addressbook`

Once you are ready with the responses for the questions in Table 20-1 and Table 20-3, you start the instance creation process just as you did when you created a new instance. When you get to the Setup Options dialog, select "A replica of an existing instance" (see Figure 20-11).

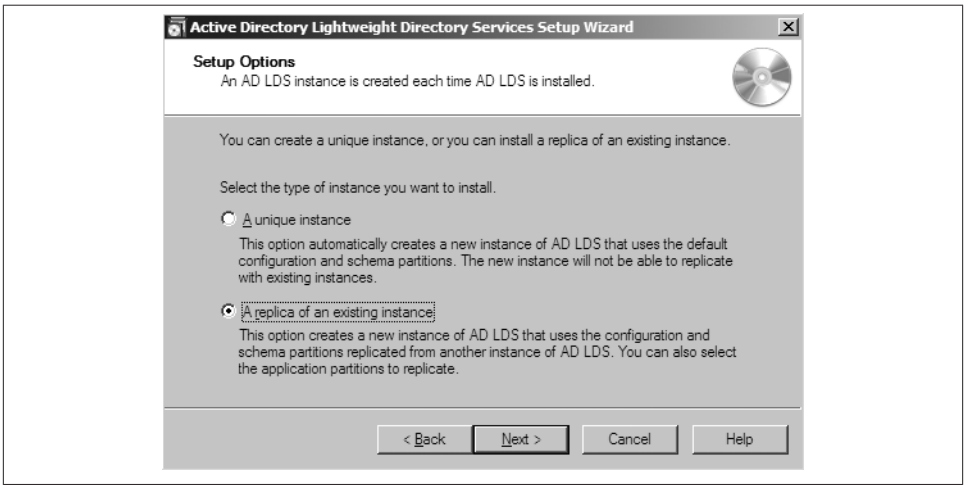


Figure 20-11. Setup Options dialog

The wizard will continue on the same path as when installing a unique instance, presenting the dialogs from Figure 20-3 and Figure 20-4. After that, the wizard will start presenting some new dialog boxes. The first new dialog allows you to specify the connection information for an AD LDS instance that is part of the configuration set you want to join (see Figure 20-12). You will need to specify both the fully qualified domain name (FQDN) of the server and the port the instance is listening on for LDAP connections.

Figure 20-12. Configuration set instance information

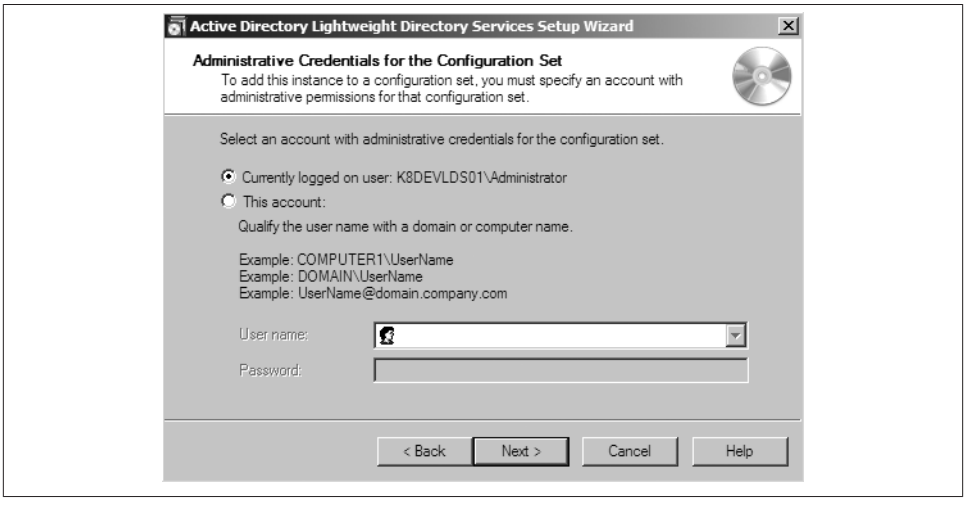


Figure 20-13. Credentials for the configuration set join

After you have specified the configuration set instance connection information, you will need to specify some administrative credentials for that configuration set (see Figure 20-13). Obviously, this is so you can actually add a new AD LDS instance into the configuration set. It wouldn't be very secure if it just let anyone add new instances.

The last new dialog that will be presented, shown in Figure 20-14, is where you get to choose which application partitions, if any, you want this specific replica to maintain a

copy of. If you refer back to Figure 20-11, you can see that you have the option to select any number of the application partitions that exist in the configuration set. In a very simple configuration set, there may only be one application partition, and that partition is shared among every replica. In a very complex configuration set, you could have any number of replicas that are or aren't replicating any number of the partitions. Again, the configuration of a complex configuration set is quite simple; monitoring and management, on the other hand, could be another story.

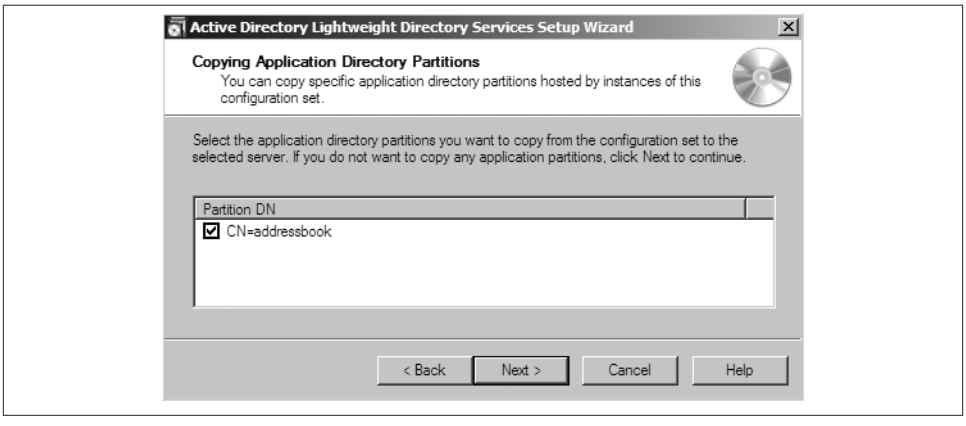


Figure 20-14. Application partitions to add to the replica

After this dialog you will return to the normal unique instance installation procedure, with the next dialog being the File Locations dialog, shown in Figure 20-6. Walk through the rest of the wizard dialogs, and after you click Finish at the end of the wizard, you will have a new replica.

AD LDS has a couple of additional installation options that we haven't mentioned yet. You have the ability to run an unattended install, as well as to install applications from media. These options can be specified when you launch the install process from the command line with *ADAMinstall.exe*. Type **`ADAMinstall /?`** or see the AD LDS help file for more information.

After you have configured a replica, you will often notice that the replica immediately has replication connections to the instance you specified in the "Joining a Configuration Set" dialog (Figure 20-12), but none of the other replicas have replication connections to the new replica. This is perfectly normal; it can take some time for the news of the new replica to circulate throughout the configuration set and the necessary replication topology to be built up with the new replica inserted into it.

If you would like to speed up this process, you can use the *repadmin.exe* command-line tool to generate the connections. Specifically, you want the */AddRepsTo* switch. Help for this switch is available if you type *repadmin /experthelp*. *repadmin* is one of the most useful tools available for determining how healthy you are from a replication standpoint and correcting any sore spots you encounter. This applies equally to AD LDS and AD.

Enabling the Recycle Bin

The Active Directory Recycle Bin, discussed in Chapter 18, is available in AD LDS configuration sets running on Windows Server 2008 R2 or newer AD LDS servers. Enabling the Recycle Bin requires a number of manual steps.

If your AD LDS configuration set was originally created prior to Windows Server 2008 R2, you must first upgrade the configuration set schema and raise the configuration set's functional level. First, upgrade the schema using this command:

```
C:\Windows\ADAM> ldifde.exe -i -f MS-ADAM-Upgrade-2.ldf -s server:port -$ ad
amschema.cat
```

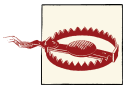
Next, using Windows PowerShell, raise the configuration set functional level:

```
Set-ADObject -Identity `
'CN=Partitions,CN=Configuration,CN=<configuration set guid>' `
-Replace @{'msds-Behavior-Version'=4} -Server server:port
```

You cannot raise the configuration set functional level until all of the AD LDS servers are running Windows Server 2008 R2 or better.

Finally, you can enable the Active Directory Recycle Bin for the configuration set:

```
Enable-ADOptionalFeature 'Recycle Bin Feature' -Scope ForestOrConfigurationSet
-Server server:port -Target 'CN=Configuration,CN=<configuration set guid>'
```



You cannot disable the Recycle Bin once it is enabled. Enabling the Recycle Bin also prevents you from rolling back the configuration set functional level. Make sure you have tested applications that take advantage of DirSync controls and/or search for deleted objects.

Tools

AD LDS introduces several new tools and other updated tools that should be familiar to most Active Directory administrators. It is worth loading AD LDS just to get these tools. Table 20-4 shows all of the tools included with AD LDS. The first four are located in the *%windir%\ADAM* directory.

Table 20-4. AD LDS tools

Filename	Description
ADAMinstall.exe	AD LDS instance installer.
ADAMsync.exe	Synchronizes data from Active Directory to AD LDS.
ADAMuninstall.exe	AD LDS instance remover.
ADSchemaAnalyzer.exe	Schema comparison tool to assist with schema differencing/synchronization. Works with AD and AD LDS.
dsdbutil.exe	AD LDS instance database management tool.
dsmgmt.exe	AD LDS instance configuration management tool.

ADAM Install

ADAM Install is the GUI tool launched by the Active Directory Lightweight Directory Services Setup Wizard menu selection. You have the option to launch the application directly to enable the advanced installation dialogs, which allow you to install an application partition from a file. You can also specify an unattended installation answer file if you would like to automate the AD LDS instance installation.

ADAM Sync

This is a command-line tool that offers basic synchronization functions to populate AD LDS information from AD. This is made available to administrators who only need a one-way feed and don't need the overhead of the Identity Integration Feature Pack (IIFP) or Microsoft Forefront Identity Manager (FIM), both of which allow two-way synchronization.

ADAM Uninstall

This tool allows you to remove AD LDS instances from a server at the command line.

AD Schema Analyzer

This is an extremely useful GUI tool that will analyze two LDAP directory schemas (live or from LDIF files) and graphically display differences between them, or "deltas." The tool can create an LDIF file with all of the deltas between the two schemas, so that you can import the LDIF file to synchronize them. The AD Schema Analyzer is helpful to use with applications that modify the schema where the vendor didn't supply an LDIF file showing the changes for you to easily review. You can also use the AD Schema Analyzer to export schema changes from a development environment to an LDIF file in preparation for making the changes in production.

AD Schema MMC Snap-in

You can take advantage of the Active Directory Schema MMC snap-in to manage the schema of an AD LDS configuration set the same way you would use it to manage the AD schema. To connect to an AD LDS configuration set, follow these steps:

1. Launch the AD Schema MMC snap-in.

The AD Schema MMC snap-in is installed but not registered by default. The first time you want to use it on a server, you must first run *regsvr32 schmmgmt.dll* from an elevated command prompt. Once you register the DLL, you can create an MMC console with the snap-in by going to Start→Run→`mmc`, and then choosing File→Add/Remove Snap-in and adding Active Directory Schema.

2. Right-click the root node on the left, Active Directory Schema, and click Change Active Directory Domain Controller.
3. Select "This Domain Controller or AD LDS instance."
4. Double-click on "<Type a Directory Server name[:port] here>" and enter the FQDN and port of your AD LDS server.

ADSI Edit

You can use ADSI Edit to connect to AD LDS instances, much like when connecting to AD domains. To connect to an AD LDS instance, right-click the ADSI Edit node on the left and click Connect To. Specify the DN of the partition to connect to and the server (and port) hosting the partition, as shown in Figure 20-15. You can optionally specify explicit credentials to connect with using the Advanced button.

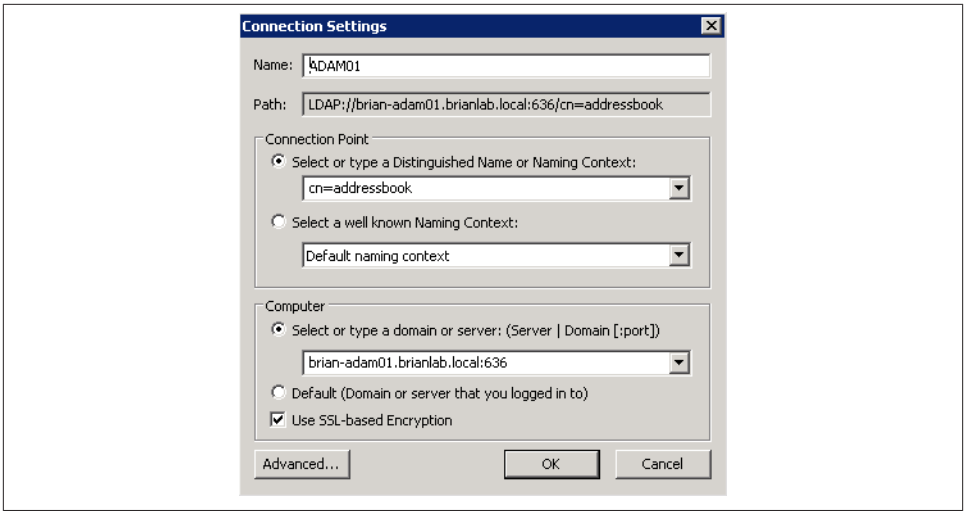


Figure 20-15. Connecting ADSI Edit to an AD LDS instance

ADSI Edit includes a special group membership editor that is useful for working with AD LDS, since it can resolve SIDs to add domain users to groups as well as allowing you to nest AD LDS security principals in AD LDS groups. Figure 20-16 shows the ADSI Edit group membership editor.

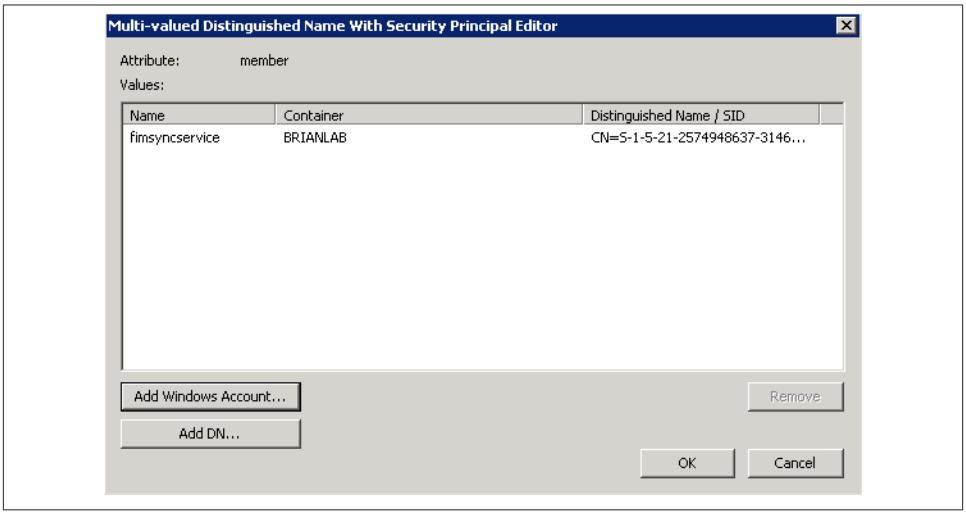


Figure 20-16. ADSI Edit group membership editor

dsdbutil

The *dsdbutil* command-line tool has a subset of the functionality present in the Active Directory *NTDSUTIL* tool and is only used for managing AD LDS. AD LDS servers can be managed with *ntdsutil*, but *dsdbutil* is specifically configured to be used with AD LDS and cannot be used with Active Directory. You can use this tool for all low-level database management tasks, such as authoritative restores, compacting files, integrity checks, etc.

dsmgmt

Like *dsdbutil*, the *dsmgmt* command-line tool has a subset of the functionality present in the Active Directory *NTDSUTIL* tool and is specifically configured to be used with AD LDS servers. You can use this tool for controlling various AD LDS configuration settings, metadata cleanup of retired AD LDS instances, and creating and deleting naming contexts.

ldifde

ldifde is a command-line tool used to import or export Active Directory objects in LDIF format. Unlike *csvde*, *ldifde* can be used to update as well as create and delete objects in AD LDS or Active Directory. LDIF is a standard format used for all LDAP-based directories and is usually the format that you receive schema update information in for applications that need schema modifications. It is important for AD and AD LDS administrators and developers to be familiar with the LDIF format in general and the *ldifde* tool in particular.

LDP

LDP is a GUI tool used to browse, search, or update AD or AD LDS. You have the ability to view the directory in a tree hierarchy or construct a standard LDAP query. LDP allows you to check in nearly any LDAP client or server control and to find deleted objects or other items that require special queries. If you are going to be managing security descriptors (ACLs) in AD LDS, you'll need to get familiar with LDP.

repadmin

The base function of the *repadmin* command-line tool is to display and manage the replication topology of a set of directory servers. But the more you use *repadmin*, the more things you'll find that it can do, from displaying object metadata to updating SPNs, displaying trusts, translating timestamps, and more. Once you have figured out everything it can do with the switches displayed when you run *repadmin /?*, you can start investigating the additional switches that are displayed when you run *repadmin /experthelp*.

The AD LDS Schema

As mentioned previously, the initial AD LDS schema is a small subset of the full Active Directory schema. You can extend the schema just as you can in Active Directory, though, so if you want to you can fully expand the schema to be the same as the AD schema. Because of this, AD LDS is a great place to test schema modifications that you want to make in Active Directory. The fact that the extensions work the same way and you can quickly destroy and recreate AD LDS instances means that you can quickly test new extensions, changing the definitions until you get exactly what you want. You can even have multiple instances on the same computer running at the same time, each with a different version of the schema so you can easily compare and contrast them. For details on working with the schema, AD, and AD LDS, see Chapters 5 and 17.

Default Security Descriptors

Quite unlike in the Active Directory schema, the base AD LDS schema `classSchema` objects do not have ACEs defined in their `defaultSecurityDescriptor` attributes. This causes objects to be created without any explicit ACEs, unless a security descriptor with explicit ACEs is specified during the object creation. The obvious benefit of this is that all of your default security is based on inherited ACEs. As you will recall from Chapter 16's discussion of Active Directory security, this makes it far easier to configure delegation. Although you are not required to follow this model for any classes you add to AD LDS, you are *strongly* encouraged to do so. You should have very great reasons to deviate from this practice.

Bindable Objects and Bindable Proxy Objects

As previously mentioned, any `objectClass` you import into AD LDS can be configured to allow bind functionality in AD LDS. The two types of bind functionality are covered by bindable objects and bindable proxy objects. A *bindable object* is an object that is actually authenticated directly by AD LDS. Bindable objects (such as users) store a password in the AD LDS database. A *bindable proxy object* is an object that is actually authenticated by Windows. AD LDS simply proxies the authentication request to Windows; an example would be a `userProxy`.

In order to make it possible to instantiate a bindable object or bindable proxy object of a given `objectClass`, you must statically associate either the `msDS-BindableObject` or the `msDS-BindProxy` class as an auxiliary class of the given `objectClass`. If you associate both classes to the given `objectClass`, `msDS-BindableObject` takes precedence.

You cannot configure existing `classSchema` objects to have either of the auxiliary classes; these classes must be specified when the class is initially defined in the AD LDS schema. The limitation is due to not being able to add mandatory attributes to an existing class definition.

Using AD LDS

Now that you have an idea of what AD LDS is and how to install it, you are probably sitting there with an empty AD LDS wondering, "What next?" This section will walk you through some common, simple tasks, including creating an application partition and populating it with some data. These examples assume you have installed an AD LDS instance with the *MS-User.LDF* and *MS-UserProxy.LDF* files. All of the examples will use LDIF files and *ldifde.exe* for making the updates, because this tool is available with every server and requires no scripting knowledge. If you are familiar with scripting or other LDAP applications, they could be used as well.

If you prefer command-line tools over GUI-based tools, point your web browser at *http://www.joeware.net* and download the *admod* and *adfind* freeware utilities. These command-line tools are specifically optimized to work with Active Directory and AD LDS and have several built-in capabilities that take advantage of special features that aren't available through ADSI.

Creating Application Partitions

Generally, the first thing you need to do with AD LDS to add data to it is to create an application partition to house that data. If you had an application partition created during the instance install, then you don't have to worry about this step unless you would like to create an additional application partition within the same instance.

AD LDS will allow you to use any container class as an application partition root. If you want the application partition root to be a `container`, `domainDNS`, `organizationalUnit`, `c`, `o`, or even a `user`, you can easily set it up. You only have to specify three pieces of data to create an application partition: the distinguished name (DN), the `objectClass`, and the `instanceType`. The `instanceType` attribute must always be `5`, so that just leaves you with two attributes you have to figure out: the DN and `objectClass`.

The distinguished name is the fully qualified name for the root of the partition. This is the value you will use any time you want to access the application partition. The `objectClass` is the type of object you want the root object to be, and the choice will impact the DN of the root. For instance, if you choose `organizationalUnit`, you know the DN will start with `OU=`.

The following example will create several application partitions with different object classes. If you want to create a single partition, only copy the application partition type you would like to use.

Create an LDIF file called *create_app_parts.ldf* with the following contents:

```
# Container-type application partition
dn: cn=AddressBook
changetype: add
objectClass: container
instanceType: 5

# Organizational Unit-type application partition
dn: ou=products
changetype: add
objectClass: organizationalUnit
instanceType: 5

# Domain DNS-type application partition
dn: dc=mycorp,dc=com
changetype: add
objectClass: domainDNS
instanceType: 5

# Organization (X.500)-type application partition
dn: o=mycorp,c=us
changetype: add
objectClass: organization
instanceType: 5
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f create_app_parts.ldf -s <server>:<ldap_port>
```

You can also use *ntdsutil.exe* to create application partitions.

Creating Containers

Creating containers in AD LDS is identical to creating containers in Active Directory. The following example will create several containers under the *AddressBook* application partition root.

Create an LDIF file called *create_containers.ldf* with the following contents:

```
# Users container
dn: cn=users,cn=AddressBook
changetype: add
objectClass: container
```

```
# User Proxies container
dn: cn=userproxies,cn=AddressBook
changetype: add
objectClass: container

# Groups container
dn: cn=groups,cn=AddressBook
changetype: add
objectClass: container
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f create_containers.ldf -s <server>:<ldap_port>
```

Creating Users

Creating users in AD LDS is similar to creating users in Active Directory. A rather obvious difference is that AD LDS users do not have the `sAMAccountName` attribute. Although this attribute could be defined and linked to the `user` class, it will not have the same special properties as it does in Active Directory, such as enforced uniqueness.

The following example will create several users under the previously created Users container.

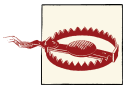
Create an LDIF file called *create_users.ldf* with the following contents:

```
dn: cn=g.washington,cn=users,cn=AddressBook
changetype: add
objectClass: user
userPrincipalName: GWashington
mail: George_Washington@cohovines.com
displayName: Washington, George
telephoneNumber: 202.555.3452
userPassword: SecureInitialPassword1!

dn: cn=j.madison,cn=users,cn=AddressBook
changetype: add
objectClass: user
userPrincipalName: JMadison
mail: James_Madison@cohovines.com
displayName: Madison, James
telephoneNumber: 202.555.3453
userPassword: SecureInitialPassword1!

dn: cn=j.adams,cn=users,cn=AddressBook
changetype: add
objectClass: user
userPrincipalName: JAdams
mail: John_Adams@cohovines.com
displayName: Adams, John
telephoneNumber: 202.555.3455
```

```
userPassword: SecureInitialPassword1!

dn: cn=t.jefferson,cn=users,cn=AddressBook
changetype: add
objectClass: user
userPrincipalName: TJefferson
mail: Thomas_Jefferson@cohovines.com
displayName: Jefferson, Thomas
telephoneNumber: 202.555.3454
userPassword: SecureInitialPassword1!
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f create_users.ldf -s <server>:<ldaps_port>
```



By default, creating enabled users in Active Directory with *ldifde.exe* requires a 128-bit SSL connection and Base64 encoding of the Unicode version of the password enclosed in quotes. See *http://support.microsoft.com/kb/263991* for more information.

For AD LDS, Microsoft enabled the `userPassword` attribute to function as a write-alias for `unicodePwd` and removed the requirement for the special formatting `unicodePwd` required. This allows your LDIF files to have clear-text passwords specified; however, you still have the SSL requirement unless you have relaxed the requirement for secure connections for password operations with *dsmgmt*.

Windows Server 2003 and newer domain controllers have the ability to also use the `userPassword` attribute instead of `unicodePwd`; you must enable that capability though the `dsheuristics` attribute.

The `userPassword` attribute can only be used for set operations in AD LDS and AD; if you need to change a password, you must use `unicodePwd`, as mentioned at *http://support.microsoft.com/kb/263991*.

Passwords are far easier to deal with using ADSI scripts, *ldp.exe*, or command-line tools, such as the previously mentioned freeware tool *admod.exe*.

Creating User Proxies

The `userProxy objectClass` is new to AD LDS. These objects are similar to users in that they represent trustees that can be authenticated; they are different in that they are only references to Windows users instead of direct representations of those users. The `userProxy` object points at a Windows user via the `objectSID` attribute on the object; that attribute can only be set on object creation.

When someone authenticates with a `userProxy`, by default, he must perform a simple bind over SSL against AD LDS with the DN or UPN of the `userproxy` object. First AD LDS accepts the `userProxy` DN/UPN and password, and then the SID is retrieved from the object and resolved to a Windows user (and domain if necessary); finally, a secure `LogonUser` API call is executed with the user's SAM name (and NetBIOS domain name if necessary) and password to authenticate the user. If the user successfully authenticates, his security token is generated and he has any access to the directory that is delegated to any of the following security principals:

- The `userProxy` object
- Any groups the Windows user is a member of
- Any groups the `userProxy` is a member of

The following example will create a UserProxy object under the previously created userProxies container.

Create an LDIF file called *create_userproxies.ldf* with the following contents:

```
dn: cn=joeproxy,cn=userproxies,cn=AddressBook
changetype: add
objectClass: userProxy
objectSID: S-1-5-21-2571958876-650952397-806722693-1108
userPrincipalName: joeproxy
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f create_userproxies.ldf -s <server>:<ldap_port>
```

In this example, the `objectSID` for a domain user is specified in the `objectSID` attribute for *joeproxy*. Any time the *joeproxy* user is used for binding to AD LDS, it will proxy that authentication through `LogonUser` to Windows and validate the credentials provided against the user with the `objectSID` in the sample.

Special considerations

The `userProxy` object can use any Windows user that can be authenticated through the trust channels of the server the AD LDS instance is running on. This includes local computer users, domain users, and trusted domain users. If you choose to use local users, you need to keep in mind that only the computer that the user exists on will be able to authenticate the user. This has obvious implications with replicated instances of AD LDS; only a single instance of the configuration set will be able to authenticate the user. Possibly this is what you are looking to do in the case of only allowing access to a single replica, but overall this would be confusing and difficult to troubleshoot if someone who wasn't familiar with this special configuration had to get involved.

Another special consideration with `userProxy` objects concerns object creation. If a Windows user's SID has already been added to the AD LDS instance as a `foreignSecurityPrincipal`, you will not be allowed to create a `userProxy` object with the same SID. The directory modification will be rejected with an "unwilling to perform" error.

Renaming Users

A common task in any address book or other application that has user information is rename operations. People are out there getting married, divorced, or just changing their names for some other reason, and they generally want to see those changes reflected in directories that contain their names.

The following example will rename a couple of previously created users.

Create an LDIF file called *rename_users.ldf* with the following contents:

```
dn: cn=g.washington,cn=users,cn=AddressBook
changetype: modify
replace: userPrincipalName
userPrincipalName: George.Washington
-
replace: mail
mail: george.washington@cohovines.com
-
replace: displayName
displayName: George Washington
-
dn: cn=g.washington,cn=users,cn=AddressBook
changetype: modrdn
newrdn: george.washington
deleteoldrdn: 1
dn: cn=j.adams,cn=users,cn=AddressBook
changetype: modify
replace: userPrincipalName
userPrincipalName: John.Adams
-
replace: mail
mail: john.adams@cohovines.com
-
replace: displayName
displayName: John Adams
-
dn: cn=j.adams,cn=users,cn=AddressBook
changetype: modrdn
newrdn: john.adams
deleteoldrdn: 1
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f rename_users.ldf -s <server>:<ldap_port>
```

Each rename consists of two LDAP operations. The first operation is to replace several of the attributes that house the old name: specifically, `userPrincipalName`, `mail`, and `displayName`. The second operation is to rename the relative distinguished name (RDN) of the object. This update forces a change of the `cn`, `name`, and `distinguishedName` attributes for `user` objects.

Creating Groups

Creating groups in AD LDS is similar to creating groups in Active Directory. As with creating users, you do not have a `sAMAccountName` attribute to be concerned with, so you can ignore that attribute.

The following example will create several groups in the previously created Groups container.

Create an LDIF file called *create_groups.ldf* with the following contents:

```
dn: cn=group1,cn=groups,cn=AddressBook
changetype: add
objectClass: group
dn: cn=group2,cn=groups,cn=AddressBook
changetype: add
objectClass: group
dn: cn=group3,cn=groups,cn=AddressBook
changetype: add
objectClass: group
dn: cn=group4,cn=groups,cn=AddressBook
changetype: add
objectClass: group
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f rename_users.ldf -s <server>:<ldap_port>
```

Adding Members to Groups

Adding members to groups with *ldifde.exe* in AD LDS is similar to the process followed for Active Directory. You have two options:

- Specify the DN of the user, group, or other object in an update to the `member` attribute of the group.
- Specify the SID of the user, group, or other object as a Base64-encoded string with the format `SID=S-1-`*`xxx-yyyy...`*.

Unfortunately, at the present time, a Windows user must be specified in the SID format unless that user has already been added to the AD LDS application partition as a `user Proxy` or `foreignSecurityPrincipal`. Once added as one of these objects, the Windows user can be referenced by the DN of the object.

The following example will add users and groups as members to the groups previously created.

Create an LDIF file called *add_ users.ldf* with the following contents:

```
dn: cn=group1,cn=groups,cn=AddressBook
changetype: modify
add: member
member: cn=george.washington,cn=users,cn=AddressBook
-
dn: cn=group2,cn=groups,cn=AddressBook
changetype: modify
add: member
member: cn=george.washington,cn=users,cn=AddressBook
member: cn=john.adams,cn=users,cn=AddressBook
-
dn: cn=group3,cn=groups,cn=AddressBook
changetype: modify
add: member
member: cn=group2,cn=groups,cn=AddressBook
member: cn=john.adams,cn=users,cn=AddressBook
member: cn=james.madison,cn=users,cn=AddressBook
-
dn: cn=group4,cn=groups,cn=AddressBook
changetype: modify
add: member
member: CN=Administrators,CN=Roles,CN=AddressBook
-
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f add_users.ldf -s <server>:<ldap_port>
```

Removing Members from Groups

Removing members from groups with *ldifde.exe* in AD LDS is similar to the process followed for Active Directory. You simply specify the DN of the user, group, or other object that is currently a member that you would like to remove.

The following example will remove members from the groups previously created.

Create an LDIF file called *remove_ users.ldf* with the following contents:

```
dn: cn=group1,cn=groups,cn=AddressBook
changetype: modify
delete: member
member: cn=george.washington,cn=users,cn=AddressBook
-
dn: cn=group2,cn=groups,cn=AddressBook
changetype: modify
delete: member
member: cn=george.washington,cn=users,cn=AddressBook
member: cn=john.adams,cn=users,cn=AddressBook
```

```
-
dn: cn=group3,cn=groups,cn=AddressBook
changetype: modify
delete: member
member: cn=group2,cn=groups,cn=AddressBook
member: cn=john.adams,cn=users,cn=AddressBook
member: cn=james.madison,cn=users,cn=AddressBook
-
 dn: cn=group4,cn=groups,cn=AddressBook
changetype: modify
delete: member
member: CN=Administrators,CN=Roles,CN=AddressBook
-
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f remove_users.ldf -s <server>:<ldap_port>
```

Deleting Objects

Deleting objects with *ldifde.exe* is generally pretty straightforward. As long as you have permission to delete the object in question and it has no children, you should have no issues with the deletion.

The following example will remove previously created group and container objects.

Create an LDIF file called *remove_ objs.ldf* with the following contents:

```
dn: cn=group1,cn=groups,cn=AddressBook
changetype: delete
dn: cn=group2,cn=groups,cn=AddressBook
changetype: delete
dn: cn=group3,cn=groups,cn=AddressBook
changetype: delete
dn: cn=group4,cn=groups,cn=AddressBook
changetype: delete
dn: cn=groups,cn=AddressBook
changetype: delete
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f remove_objs.ldf -s <server>:<ldap_port>
```

Deleting Application Partitions

Programmatically deleting an application partition in AD LDS is not an intuitive process. When you created a partition, you specified the DN, `objectClass`, and `instanceType`, and that was all that was required. When you need to delete an application partition, you have to look at the objects in the `cn=partitions,cn=configuration,cn=`*`<instance GUID>`* container. Locate the object with the `nCName` that matches

the application partition you want to delete and copy the DN into your LDIF file that has the delete operations.

The following example will delete the `cn=AddressBook` application partition we created earlier.

Create an LDIF file called *remove_app_part.ldf* with the following contents:

```
#>nCName: CN=AddressBook
dn:
  CN=45d599d5-9bd4-41be-9604-b8209db3f866,CN=Partitions,
  CN=Configuration,CN={92B82A0E-CEC4-4720-9035-D0CA9632C20E}
changetype: delete
```

Once you have created the LDIF file, run the following command:

```
> ldifde -i -f remove_app_part.ldf -s <server>:<ldap_port>
```

You can also use *ntdsutil.exe* to delete application partitions.

Controlling Access to Objects and Attributes

We discussed delegation of access to Active Directory in Chapter 16, and all of the concepts discussed there also apply to AD LDS. There are, however, two critical differences. The first, as mentioned briefly earlier in this chapter, is that unlike AD, AD LDS does not grant any read access to the directory out of the box. This means that as the administrator, you need to either grant full read access by nesting users and groups in the Readers group for your instance, or delegate permissions more granularly. The second difference is that the ACL editor and Delegation of Control Wizard you're used to using are not available for use with AD LDS. You must instead take advantage of the ACL editor in the LDP utility, which is unfortunately not nearly as user-friendly.

You can access the LDP ACL editor by following these steps:

1. Launch LDP by going to Start→Run→ and entering **`ldp.exe`**.
2. Connect to the server from Connection→Connect. Enter the FQDN and port of the AD LDS server hosting the application partition for which you want to manage security.
3. Bind to the instance from Connection→Bind (Ctrl-B). You can either select "Bind as currently logged on user" to use your domain credentials, or specify alternate credentials.
4. Load the application partition hierarchy by selecting View→Tree (Ctrl-T) and choosing your application partition.

5. Right-click the object you want to edit the ACL for and click Advanced→Security Descriptor. Click OK.

If you need to edit auditing settings, you'll need to check the SACL checkbox in the preceding dialog.

6. Click in the DACL listbox to enable the Add ACE, Delete ACE, and Edit ACE buttons at the bottom of the dialog.

Figure 20-17 shows an example where we are granting a group called "Property Set Read – MyCorp Personal Information" (the trustee) read property rights (access mask) to the MyCorp Personal Information property set (object type) for all users (inherited object type). This is being applied to the People OU and will be inherited by all users in that OU structure.

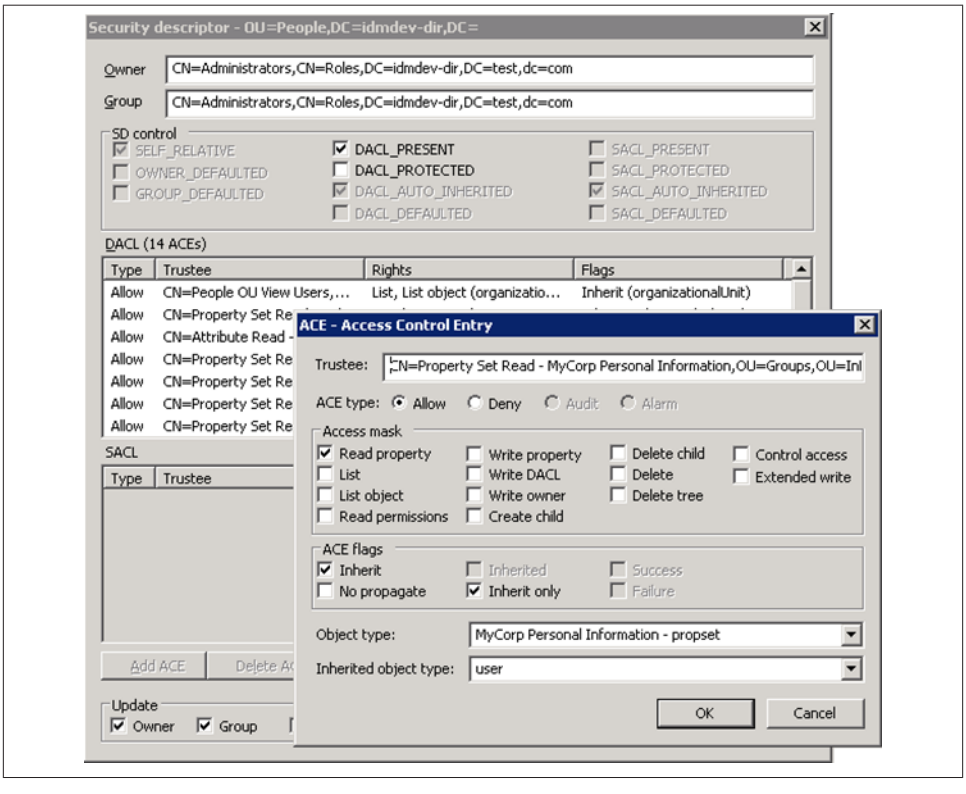


Figure 20-17. Granting Read Property access to a group

In Figure 20-18, we are granting a group called People Full Control (trustee) most rights (access mask) to all of the users (object type) in the People OU.

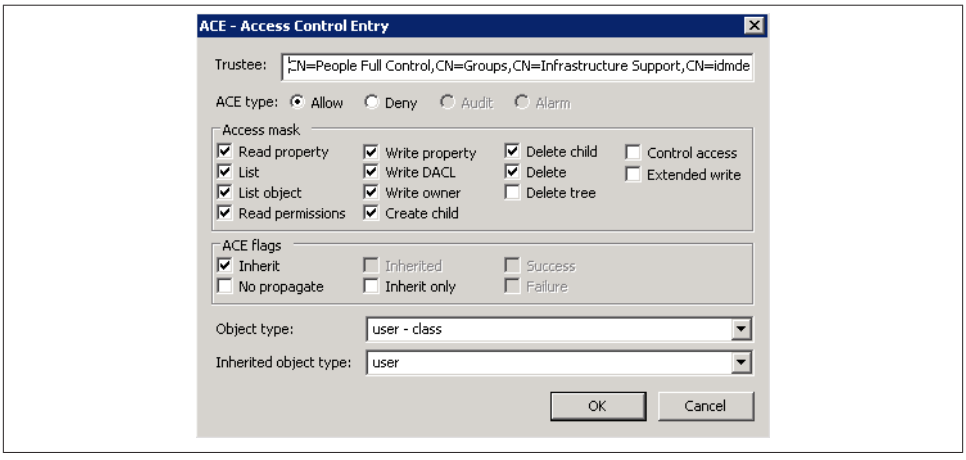


Figure 20-18. Granting access to users in an OU

Summary

AD LDS is an extremely exciting product that is certainly going to find heavy use for both Microsoft and other applications in the enterprise environment. The flexibility it offers allows Active Directory administrators to say "We can do that" more often now in order to support line-of-business applications and other functions that they would never have allowed near their domain-based Active Directory networks. Knowledge and understanding of AD LDS will become more and more important to all Active Directory administrators going forward as more and more applications start to leverage it.

In this chapter, basic AD LDS concepts, as well as some more advanced concepts, were discussed and explored. We walked through the installation of the AD LDS server role as well as the installations of a standard unique instance and a replica instance, with screenshots and tables illustrating decision points. We also introduced the new and updated tools that ship with AD LDS. Finally, many *ldifde.exe* examples showing most aspects of creating and deleting objects and partitions were provided and discussed to help a new AD LDS administrator quickly get up and running with a new AD LDS instance.

CHAPTER 21

Active Directory Federation Services

As services have transitioned from the corporate datacenter to platforms hosted externally in the "cloud," the need for a mechanism to extend authentication has evolved substantially. Microsoft's answer to this is Active Directory Federation Services (ADFS). ADFS is a standalone service that carries the Active Directory brand, so naturally it is expected that Active Directory administrators will be prepared to deploy and manage ADFS.

In reality, ADFS is a skillset in its own, so we'll take the opportunity in this chapter to introduce the service and discuss the basics of deploying and configuring it. If you've ever done any web development, you'll probably find many of the concepts of ADFS and identity federation in general to be far more familiar than others who are coming in with a purely infrastructure-based background.

No doubt, either way you'll find many of the fundamental concepts of identity federation to be strikingly similar to concepts that have existed since the early days of Active Directory (and even earlier with NT domains). If you explore the federated identity space, you'll discover that this space is crowded and numerous vendors have solutions to this problem. Fundamentally, many of these solutions are functionally very similar. We'll limit our discussion in this chapter to ADFS, but many of the concepts we discuss apply broadly across the market.

Introduction to Federated Identity

Fundamentally, the idea behind identity federation is that one party (the *identity provider*, or IdP) will be able to assert that a user is who she says she is to a second party (the *relying party*, or RP) with nothing more than a lightweight trust relationship established between the IdP and the RP. The term "trust" is important in this description as there is no way for the RP to actually know that the IdP is providing an entirely factual answer.

To make this example less abstract, consider a scenario where an organization (Coho Vineyard) outsources its employee payroll processing to a third party (Fabrikam Payroll). In order for employees to be able to access their payroll and tax information online, they need a way to log into the Fabrikam Payroll website that is hosted in Fabrikam's datacenters.

Figure 21-1 outlines this scenario. In Figure 21-1, the users exist in Coho Vineyard's Active Directory. Coho Vineyard has deployed an ADFS environment and is the identity provider in this case. Coho Vineyard and Fabrikam Payroll have worked together to establish a *federation trust*. In this case, the Fabrikam Payroll website is the relying party.

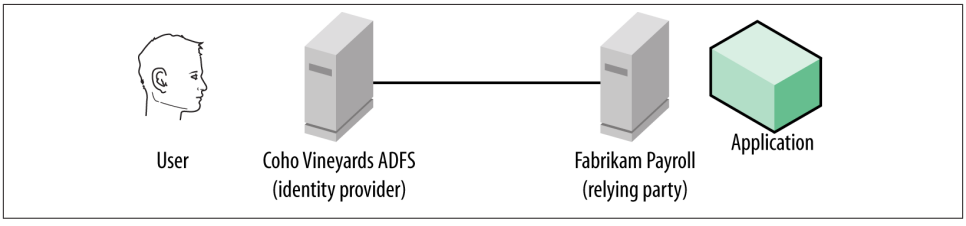


Figure 21-1. Federation overview

How It Works

At the end of the day, there are a couple of protocols that are spoken between partners in an identity federation scenario. No matter where you go, chances are you'll be dealing with either Security Assertion Markup Language (SAML, pronounced "sam-il") or WS-Federation (often abbreviated WS-Fed).

For SAML and WS-Fed, the conceptual semantics of how they work is the same, but the formatting of the data is different. In both cases, the IdP generates an Extensible Markup Language (XML) snippet that contains a series of *claims* about the user in question. Claims are simply attribute/value pairs. Common examples are data points such as the user's name, email address, employee number, and so forth. This set of claims and the supporting data in the XML comprise the user's *token* for access to the RP. The RP uses the claims in the token to authorize the user and provide the necessary functionality in the application.

In order to guarantee that the token was issued by the IdP and that it hasn't been tampered with, the IdP digitally signs the XML with its *token signing certificate*. The RP has a copy of the IdP's token signing certificate public key. When the RP receives a token from the IdP, it uses the public key to confirm that the token is valid.

Now that we've discussed the token that the IdP issues, let's look at an end-to-end example whereby a user wants to access the Fabrikam Payroll website, as shown in Figure 21-2.

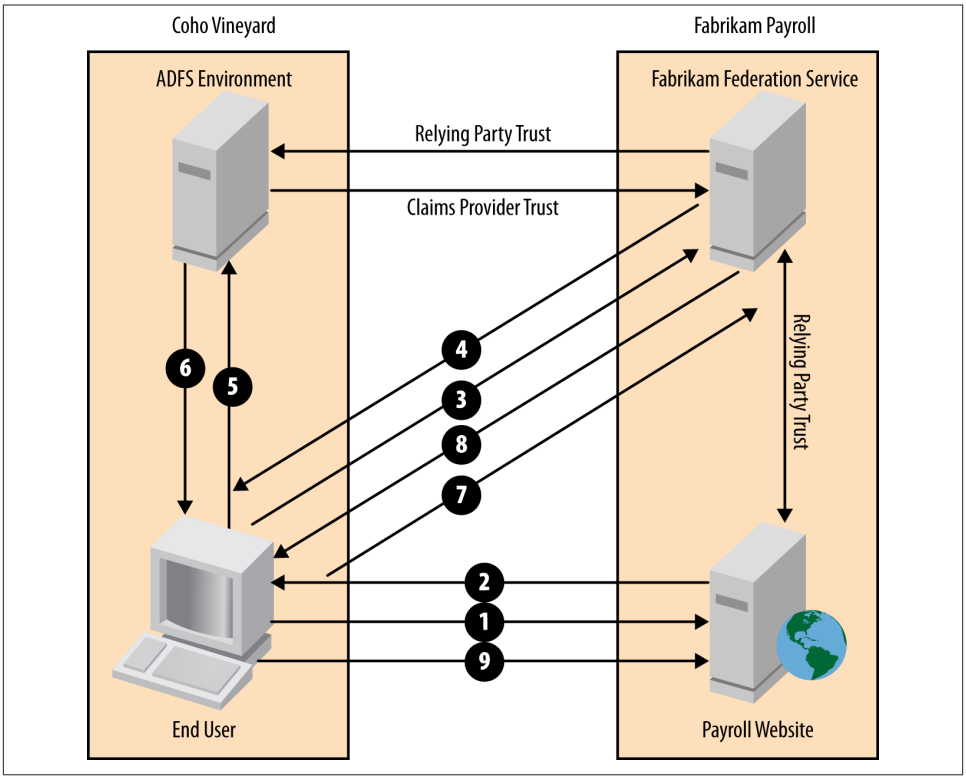


Figure 21-2. Federated application access workflow

In this scenario, the following steps occur for a user to access the Fabrikam Payroll website:

1. The user browses to the Fabrikam Payroll website with his web browser, directing it. At the Fabrikam Payroll website, the user attempts to log in and the application discovers that the user must be logged in via federation.
2. The website issues an HTTP redirect to the user's browser directing it to visit the Fabrikam Federation service URL for authentication.
3. The user's browser accesses the Fabrikam Federation service.

The Fabrikam Federation service could be an ADFS environment, or it could be running any one of a multitude of other federation products available on the market.

4. The Fabrikam Federation service looks up the URL for the Coho Vineyard IdP and issues an HTTP redirect to the user's browser.
5. The user's browser accesses the Coho Vineyard ADFS server.
6. The user authenticates to the ADFS server and is presented with a token that is signed with the ADFS server's token signing certificate. The ADFS server issues an HTTP redirect to the user's browser, directing it back to the Fabrikam Federation service.
7. The user's browser presents the token to the Fabrikam Federation service.
8. The Fabrikam Federation service validates the token and then issues the user a new token for the Fabrikam Payroll website that is signed with Fabrikam's token signing certificate. Finally, the Fabrikam Federation service issues an HTTP redirect to the user's browser pointing it back to the Fabrikam Payroll website.
9. The user accesses the Fabrikam Payroll website and presents the token issued by the Fabrikam Federation service. The website validates the token, authorizes the user, and grants him access to the application.

You're probably wondering how all this could possibly work or be efficient with this many redirects and round-trips. Perhaps surprisingly, the process works extremely well and is quite efficient. There are a couple of things that might not be immediately clear from the process flow described here.

The first is step 2. Why does the Fabrikam Payroll website redirect the user to the Fabrikam Federation service versus directly to Coho Vineyard's ADFS service? The answer here is in the semantics of building applications for federated identity. There are two trust relationships that come into play in Figure 21-2. In the case of the website or applications in general, the application can only trust one federation service. The federation service, however, can trust many other IdPs. Since many customers use the Fabrikam Payroll website, the website has a trust with Fabrikam's federation service, and then in turn Fabrikam's federation service trusts each customer (Coho Vineyard and so forth).

In some cases, the website or application also functions as the federation service. In this case, the trust relationship would be directly between the customer's ADFS server and the relying website. This removes a couple of hops and is a bit simpler, but it is functionally identical.

With that piece of information in mind, it may become a bit clearer why we need to make two round-trips to the Fabrikam Federation service. In step 6, the user is presented with a token signed by Coho Vineyard's ADFS server. Since the trust relationship between Coho Vineyard and Fabrikam Payroll is between the federation services, the user

must present the signed token back to Fabrikam's federation service (step 7). Fabrikam's federation service will then validate the token, and if everything checks out, it will issue a *new* token that is signed with Fabrikam's certificate (step 8). Since the Fabrikam Payroll application only trusts Fabrikam's federation service, the website would not be able to validate the token if it were signed by Coho Vineyard's ADFS server.

Before moving on, we'll leave you with a comparative piece of food for thought. If you're an Active Directory administrator, you're undoubtedly familiar with Kerberos (Chapter 10). The fundamentals of how identity federation works, as we just discussed, are very similar to Kerberos. Functionally, you can think of the IdP as a domain controller and the RP as an application or server (e.g., a file server). When a user accesses the file server, she must present the file server with a service ticket (the federation token) in order to access the service. The service ticket is encrypted with the service's secret to ensure that it isn't tampered with (similar to the signing of the token from the federation service).

SAML

Security Assertion Markup Language (SAML) 2.0 is one of the two primary competing standards for passing federation tokens between environments. In the United States, you'll find SAML to be extremely prevalent in education and government; it is also used in many third-party applications and cloud services. For the external cloud services, supporting SAML is the logical choice for achieving a broad range of compatibility with potential customers.

You can learn quite a bit about SAML on Wikipedia, of all places. We found the white-paper "How to Study and Learn SAML" to be extremely helpful as well. Finally, the specifications for SAML are available online.

As you read documentation on SAML, you'll probably notice that the documents discuss the term *assertion* at great length. In SAML, an assertion is functionally equivalent to the term "token" that we use in this chapter.

WS-Federation

WS-Federation is the protocol Microsoft has chosen to adopt in its applications and software development kits (SDKs). In fact, the first version of ADFS, only knew how to speak WS-Federation. This was an issue that greatly impacted the adoption of ADFS, as large portions of the market were inaccessible without a service to convert between SAML and WS-Fed.

If you work with developers who are building .NET applications using Windows Identity Foundation (WIF) or the classes included in the .NET 4.5 Framework, those applications will generally speak WS-Fed in their interactions with the IdP.

Understanding ADFS Components

Now that we've spent some time exploring the semantics of identity federation at a conceptual level, we'll spend the rest of this chapter diving into deploying and configuring ADFS. In this section, we'll first discuss the server roles that comprise an ADFS topology. After that, we'll take a look at some common deployment topologies for environments of different sizes.

The Configuration Database

The first decision you'll need to make before you begin deploying ADFS is where to store the ADFS configuration. There are two places where ADFS can store its configuration—: on a Windows Internal Database (WID) instance that's replicated to each federation server, or in a shared SQL Server environment.

Both options have pros and cons. The WID database is extremely easy to deploy (WID is included with Windows) and comes with no additional licensing costs. WID also removes the single point of failure in a distributed ADFS environment. With WID, each federation server contains a copy of the configuration database. One federation server (usually the first federation server deployed) is designated as the "primary" server. All configuration changes are made on the primary server, and these changes are replicated to all of the "secondary" federation servers every five minutes.

The WID database also has a few limitations. First, it has limitations in terms of performance and capacity in extremely large ADFS deployments. Additionally, storing the configuration database on WID removes support for SAML artifact resolution and token replay detection.

SAML artifacts are a protocol feature specific to SAML (versus WS-Fed). SAML artifacts allow SAML tokens (assertions) to provide a pointer to a piece of data that can be independently retrieved, rather than embedded in the SAML assertion. Since the request to retrieve the artifact will come at a later time, and the request might go to a different federation server, the artifact must be stored in the configuration database to make it globally available.

Token replay detection ensures that a token that is issued by ADFS can't be reused. ADFS stores information about each token it issues in the configuration database to ensure that a token isn't subsequently reused. This feature also requires that the configuration database be stored in a shared SQL Server environment versus in WID.

Finally, by using a SQL Server environment, you can take advantage of SQL high-availability features such as clustering and mirroring. These features can be useful in geographic redundancy scenarios, for example. We'll discuss geographic redundancy in the section "ADFS Topologies".

Federation Servers

The federation server is the heart and soul of an ADFS environment. Federation servers are capable of acting as both IdPs and RPs, depending on the scenario. All of the protocols ADFS speaks (e.g., SAML and WS-Fed) are emitted by the federation server as well.

Proper security (both physical and logical) of the federation server is important. Federation servers are just like domain controllers in that they control access to applications and services. Anyone with access to a federation server could impersonate any user in the organization by creating a false token that is signed with the federation server's token signing certificate. To a relying party, the token would appear authentic and valid and would be treated just like a token that wasn't fraudulently issued in this manner.

Federation Server Proxies

The federation server proxy is an optional feature that is used to provide a layer between your federation servers and the Internet. Functionally, there's nothing you can't do with ADFS if you don't deploy proxies, but there are some scenarios that are much easier to handle with proxies in the mix.

The main scenario that federation server proxies enable out of the box is the ability to have different authentication mechanisms for internal and external users. Internally, when users connect to your federation server on the corporate network to obtain a token, you can rely on Windows Integrated Authentication (Kerberos and NTLM) to pass the user through to ADFS without any additional prompts. Externally, however, your users may be using their home PCs or connecting at a coffee shop, for example. In this case, you will likely want to present a friendly HTML form to authenticate the user.

Federation server proxies also insulate external clients from the relatively sensitive data on the federation server—the private keys that are used to sign tokens. This provides a level of security that many organizations appreciate.

Some organizations opt to use a reverse proxy other than the federation server proxy to publish their ADFS servers, and others opt to publish their federation servers directly on the Internet. Microsoft recommends the use of federation server proxies throughout its documentation, and we'll show you how to deploy them later in this chapter.

ADFS Topologies

There are a number of different topologies for deploying ADFS, ranging from small and simple to large, complex, and highly redundant. We'll look at a few common options so that you're familiar with them and then move on with a walk-through of a relatively simple deployment that we'll use for the rest of this discussion.

As you review the designs we present in this section, think about your environment's requirements for scalability and, more importantly, high availability. As you begin federating with services hosted by third parties, your environment (in this case, ADFS) will still be critical to maintain access to those applications and services. If your ADFS environment is down, your users will not be able to access the cloud solution.

This is a trap that many organizations fall into as they briefly rejoice in the thought that they will not need to deliver high service levels for services that they outsource. While you will no longer be on the hook for maintaining the outsourced services, you will instead be on the hook for ensuring high availability for your ADFS environment, which governs access to the hosted service.

Single federation server

The simplest topology, suitable for testing and development, involves the deployment of a single federation server. In this scenario, you simply install the ADFS server role on a server and then use the configuration wizard (as described later in this chapter) to deploy a federation server using the Windows Internal Database. Outside of testing and development, the utility of this topology is likely quite limited.

Single federation server and federation proxy

In this scenario two servers are be required, as shown in Figure 21-3. One server functions as the federation server and host of the configuration database, and the other server functions as the federation server proxy and publishes ADFS to the Internet.

This topology is perfectly suitable for an organization that needs to federate with one or more partners and applications but does not have a need for redundancy at the server level. ADFS's architecture makes it relatively easy to scale out this topology later to support multiple federation servers and federation server proxies.

You could optionally deploy the configuration database on a standalone SQL server, but unless you have a need for SAML artifact resolution or token replay detection, it is unlikely that this will be necessary.

Load-balanced ADFS servers

As soon as you want to achieve any redundancy with ADFS, you'll need to bring load balancing into the mix. This will likely require that you work closely with your colleagues on the network team to complete your deployment. As load balancing configurations go, the configuration for ADFS is quite simple. You will simply need to configure Layer 4 load balancing of TCP ports 80 and 443. There is no need for server affinity or SSL offloading unless you have a specific need outside of the basics. You will, however, want to configure appropriate server health monitoring on the load balancer.

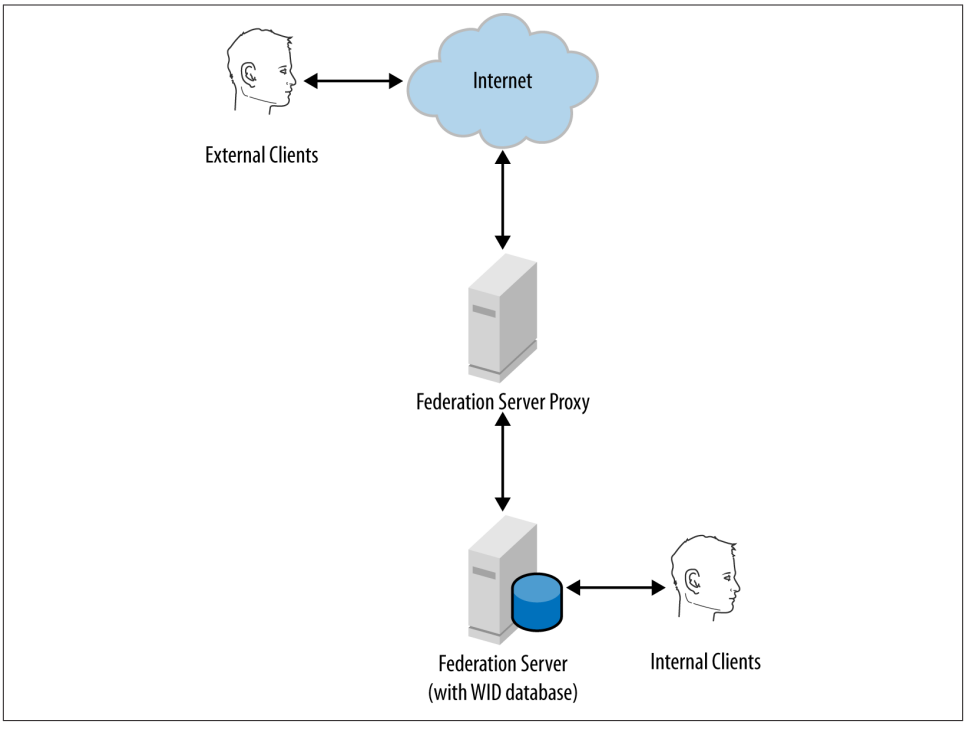


Figure 21-3. No high-availability deployment scenario

While it is possible to use the Windows Network Load Balancing (NLB) service that comes with Windows in lieu of a hardware load balancer, we strongly recommend that you elect to use a hardware load balancer instead.

Figure 21-4 shows a standard topology with load balancers in the mix. In this case, we have a pair of federation server proxies that are load balanced, and those proxies are connecting to a set of federation servers that are independently load balanced. We've opted to continue using the Windows Internal Database for configuration storage here. You could, however, deploy the database on a SQL Server cluster if necessary to support the scale of your deployment or because of a need for features such as SAML artifact resolution or token replay detection. As your requirements grow, you can easily add additional federation servers and federation server proxies behind the load balancers.

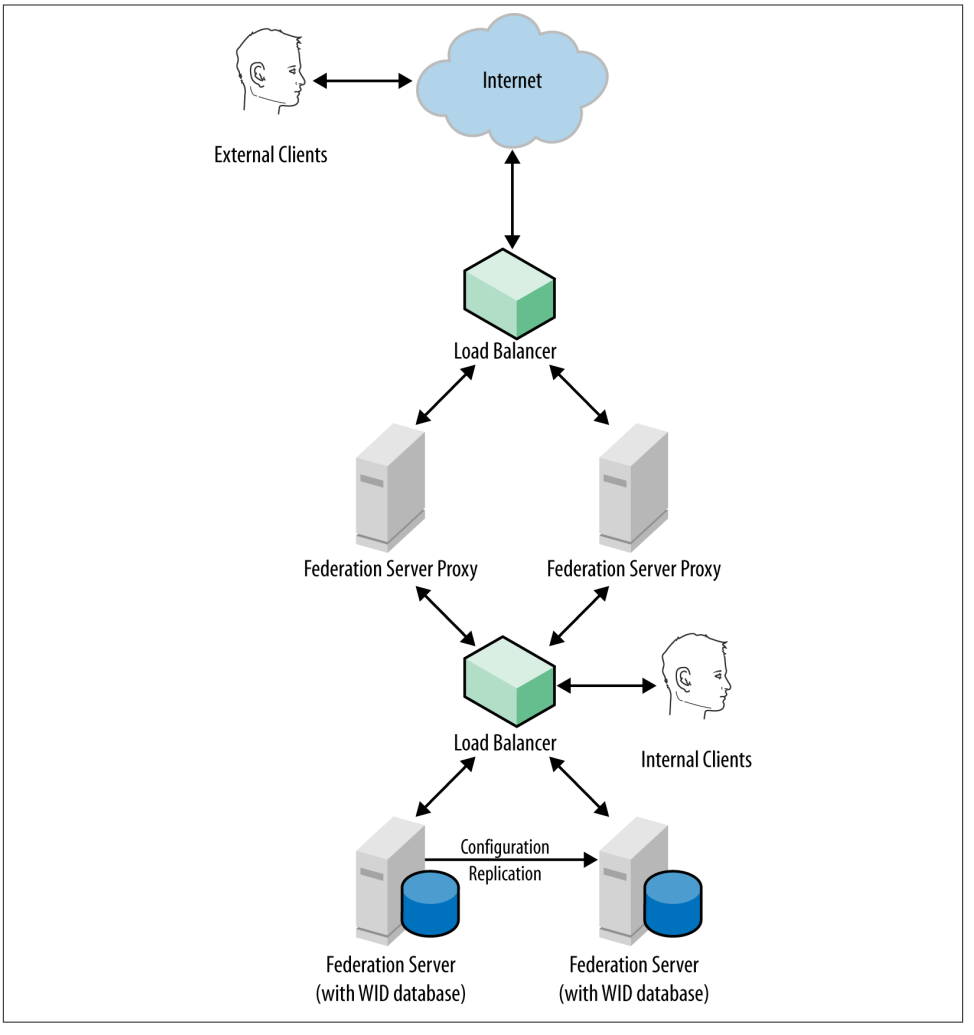


Figure 21-4. Load-balanced topology with WID

While the diagram shows separate load balancers, depending on your network topology, you may simply need to configure two virtual IPs (VIPs) on one load balancer. Work with your network team to determine the best architecture.

Geographically redundant ADFS servers

The last and most complex solution involves placing ADFS servers in multiple datacenters to ensure the highest levels of availability. In this case, you will need to deploy a number of complex technologies to ensure that one DNS name (your ADFS service name) resolves to servers in multiple datacenters worldwide, and that users are directed to the datacenter that is closest to them.

In order to achieve the configuration shown in Figure 21-5, you will need a geographic load balancing service in addition to load balancing in each datacenter. There are various services and products that offer this capability, usually by acting as the DNS server for the services that are behind the global load balancer. One solution we are familiar with is the F5 Global Traffic Manager (GTM). The GTM is just one solution in a crowded market, so you should once again plan to work closely with your network team to determine the best solution.

In addition to the global load balancing solution, you will need to employ SQL Server's mirroring capabilities to replicate the configuration database into each datacenter that hosts an ADFS environment.

Deploying ADFS

Now that we've taken a look at the ADFS server roles and some common topologies, we'll go ahead and walk through the deployment and initial configuration of an ADFS environment. The environment we'll configure is shown in Figure 21-3. Specifically, we'll install a standalone federation server as well as a standalone federation server proxy. We'll use the Windows Internal Database for our configuration store.

Before we go too far, we have to make one configuration decision—what should the FQDN of our environment be? This will need to be a name that is resolvable both internally and externally. This is the URL that users will be redirected to in order to authenticate and that relying parties will federate with. Technically speaking, the URL you pick has no impact as long as it can be resolved internally to the federation server and externally to the federation server proxy, but keep in mind that this is a URL you're not going to be able to change later.

A few common examples we've seen are:

- *sts.cohovines.com*

STS stands for Security Token Service. STS is another term for IdP that is used by some products in the federation space.

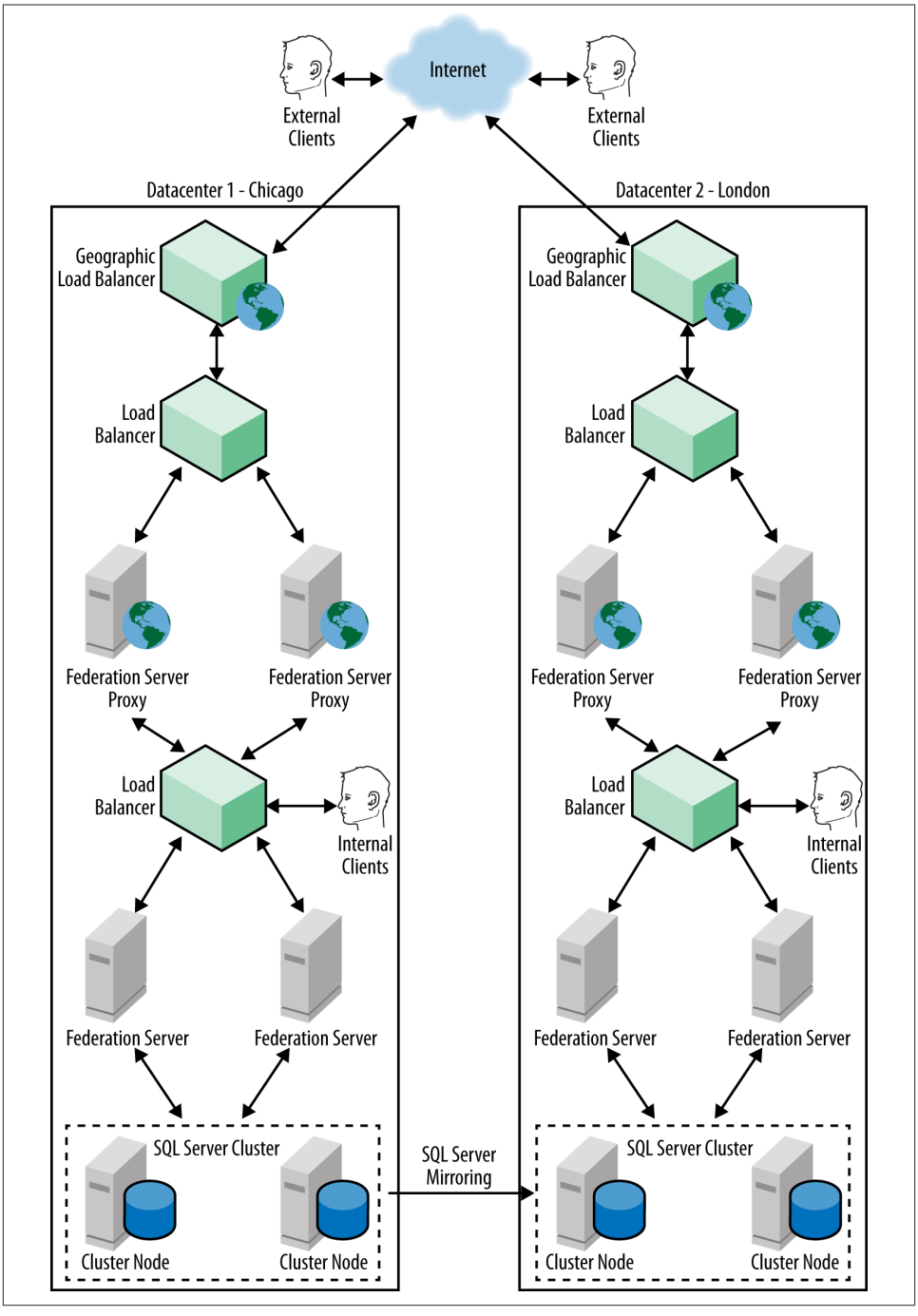


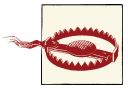
Figure 21-5. Geographically load-balanced server farms

- *fs.cohovines.com*
- *auth.cohovines.com*
- *login.cohovines.com*

For the rest of this chapter, we're going to use *sts.cohovines.com* as the name for our federation service.

Federation Servers

The first step to an ADFS deployment is deploying the federation server. If you're running Windows Server 2008 R2, you'll first need to download and install ADFS.



The version of ADFS included with Windows Server 2008 R2 is ADFS 1.0. ADFS 1.0 is markedly different from ADFS 2.0, so you'll want to make sure you're deploying the latest version.

If you're running Windows Server 2012, ADFS 2.1 is that the latter is included out of the box as an optional server role. The only difference between ADFS 2.0 and ADFS 2.1 is that the latter includes the ability to send Dynamic Access Control (DAC) claims embedded in your Kerberos ticket as ADFS claims. Regardless of which version of ADFS you're running, the directions in this chapter stand.

Both the downloadable setup for Windows Server 2008 R2 and the out-of-the-box Windows Server 2012 role setup are smart enough to install ADFS's prerequisites. Both setup wizards will ask if you'd like to install a federation server or a federation server proxy. You should stick with the default of Federation Server for this step.

Certificates

ADFS requires a number of SSL certificates in order to function. At a minimum, you need to provide a certificate for the FQDN of your federation service (*sts.cohovines.com* in our case). Make sure the certificate is trusted by the clients that will be accessing the service.

In order to get an SSL certificate, you'll need to generate a *certificate signing request* (CSR) and submit the request to a *certification authority* (CA). There are a number of ways to generate the CSR. We typically use the free utility from DigiCert, a reputable commercial CA with excellent customer service. You can download this utility from *https://www.digicert.com/util/* and run it directly on the federation server. The CSR the utility creates can be submitted to any internal or commercial CA.

To create the CSR, launch the utility on your federation server and click Create CSR. Fill in the form as appropriate for your environment. The Common Name field should

be the FQDN of your federation service, as shown in Figure 21-6. Once you receive the CSR, you can submit this to your CA.

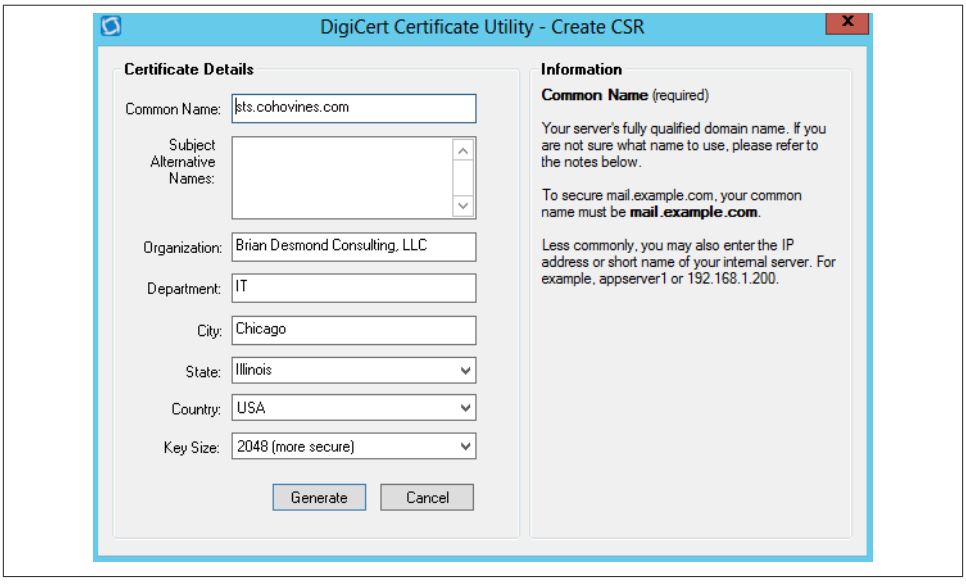


Figure 21-6. DigiCert CSR creation wizard

When you receive the certificate back from the CA, use the Import button at the bottom of the utility to import the result. When this completes, you should see the certificate with a green checkmark, similar to Figure 21-7. If you are deploying a federation server farm or federation server proxies, you can use the Export button at the bottom of the screen to export the certificate to a PFX (Personal Information Exchange) file and then import it on the other servers.

There are two other types of certificates on the federation server—the token signing certificate and the token encryption certificate. When you install a federation server, ADFS will generate self-signed certificates for these purposes. We'll discuss these in more detail in the section "Service configuration" on page 626.

Configuring ADFS

Once setup completes, you can launch the Active Directory Federation Services MMC snap-in. Inside the console, click the link to launch the AD FS Federation Server Configuration Wizard to get started. Before you proceed, take a moment to review the

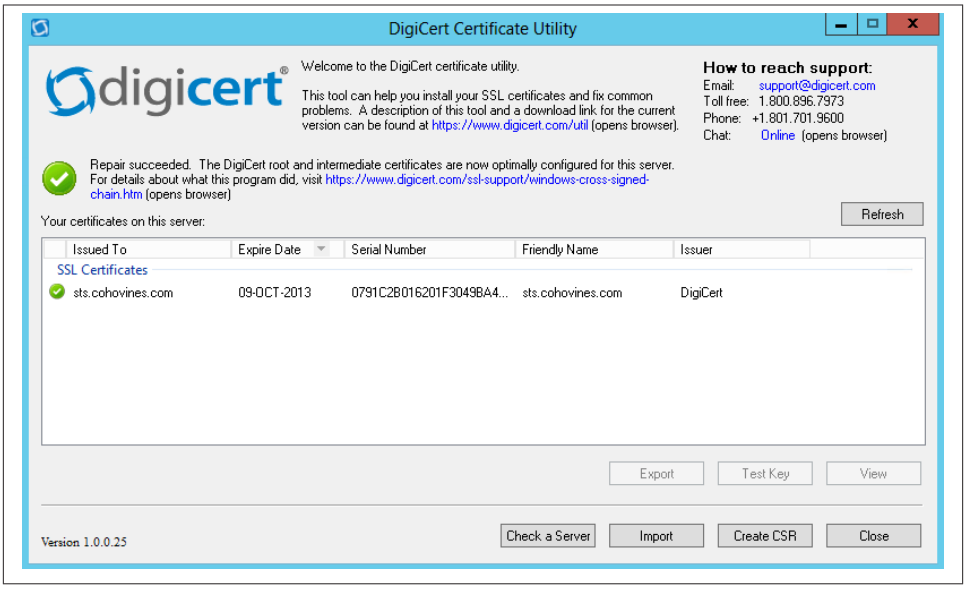


Figure 21-7. DigiCert certificate

sidebar "ADFS Configuration Wizard Permissions" to make sure the wizard won't fail when you get to the end.

ADFS Configuration Wizard Permissions

The configuration wizard will fail if you're not running as a Domain Admin. This is because the wizard has to publish a key sharing container to the Program Data container in Active Directory.

This container is used for federation servers to securely share private keys, specifically the token signing and token encryption certificates. If you can't run the configuration wizard as a member of the Domain Admins group, you have a couple of options.

The first option is to have a domain admin run the *Set-ADFSCertSharingContainer* PowerShell cmdlet in advance. This will create the container in Active Directory and set the permissions for the ADFS service account to manage the container.

The second option is to precreate a container called Microsoft under Program Data, and then inside the Microsoft container create a second container called ADFS. Delegate the user running the ADFS configuration wizard full control of the ADFS container in Active Directory.

Figure 21-8 shows a published key sharing container in Active Directory from a successful ADFS installation. If you have multiple ADFS farms in your domain, you will find multiple ADFS containers.

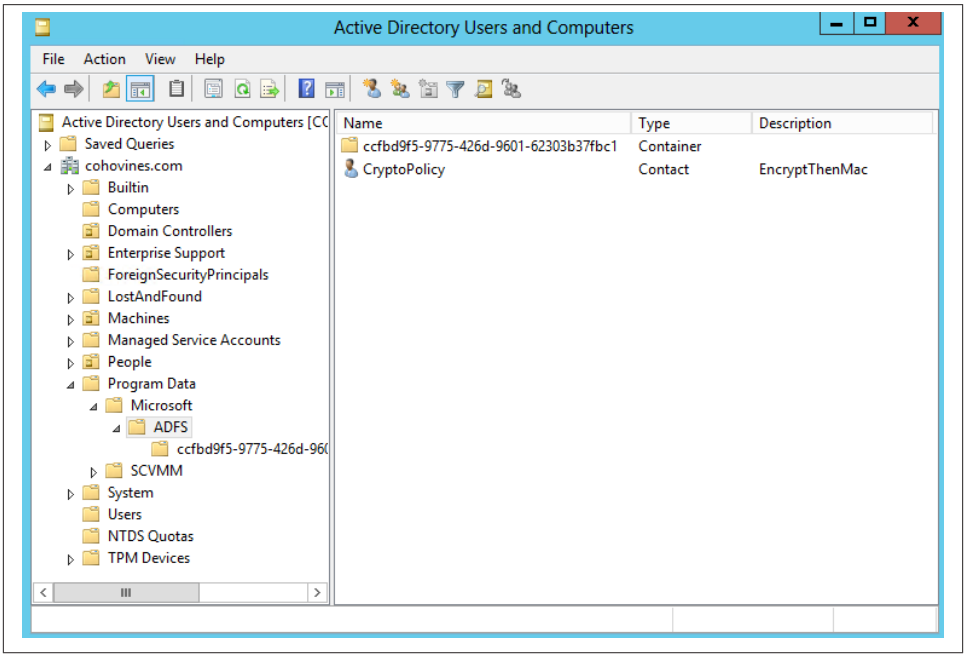


Figure 21-8. Key sharing container

If you are going to use a SQL Server database for the configuration database in lieu of WID, you'll need to use a command-line utility to configure ADFS. That utility is called *fsconfig.exe* and is available in *%windir%\ADFS*.

The first thing you'll be asked, as shown in Figure 21-9, is whether you want to deploy a new federation service or join an existing farm. Select the option "Create a new Federation Service."

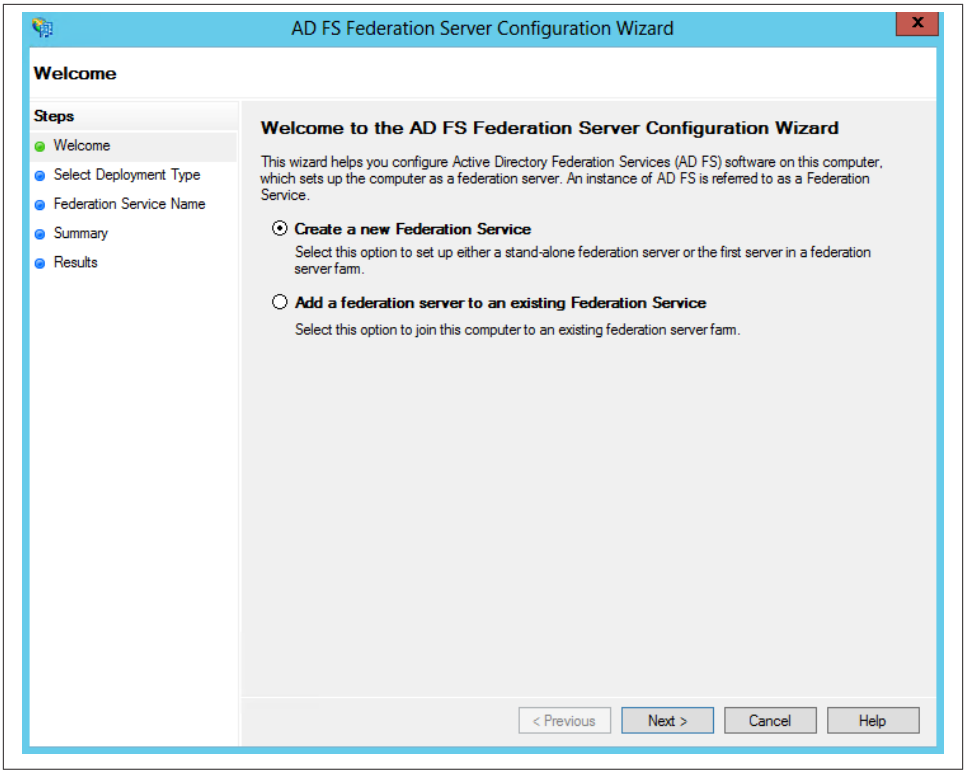
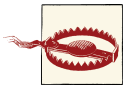


Figure 21-9. Create or join a federation service

Next you'll be given the option to create a new federation server farm or to install a standalone server, as Figure 21-10 shows. Even if you don't plan to add additional federation servers, we recommend that you opt to create a one-server farm.



There is no transition path available to move from a standalone server to a federation server farm later.

ADFS also requires a service account in order to run in a farm environment. This service account should be a normal Active Directory user account, with the exception that it should have the service principal names (SPNs) for the federation service listed on it. These are generally similar to `http/sts` and `http/sts.cohovines.com`.

The configuration wizard will prompt you to choose an SSL certificate to assign to the Internet Information Services (IIS) website that ADFS uses. Be sure to choose the certificate that matches your federation service name, as shown in Figure 21-11. If you need to change this, you can do so easily later in the IIS management console.

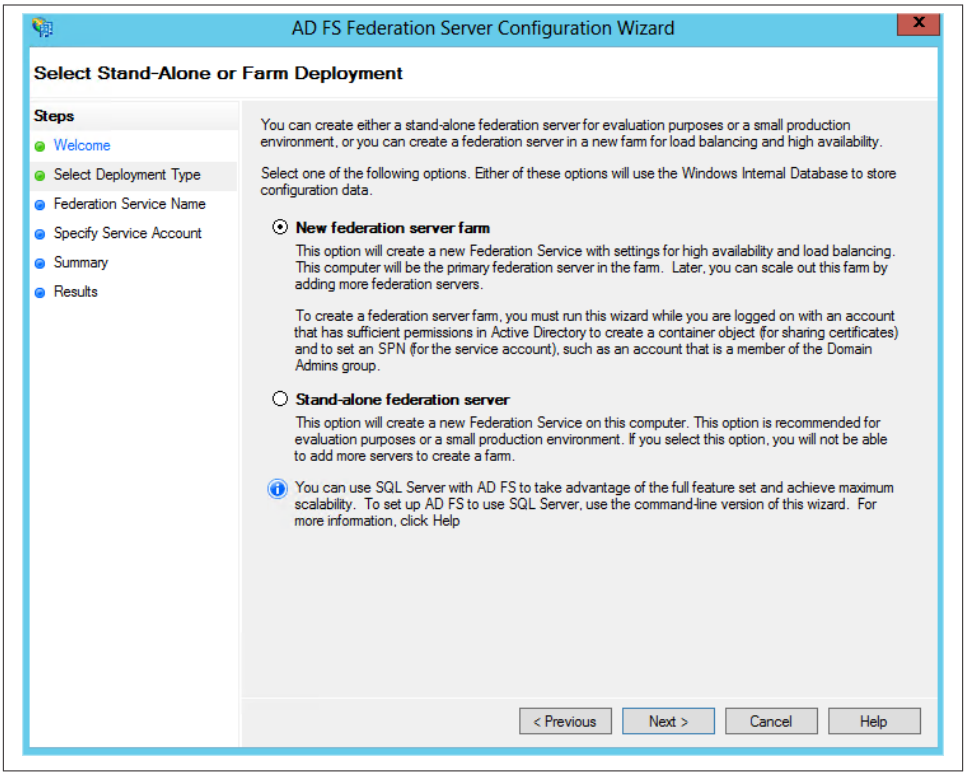


Figure 21-10. Create a standalone server or server farm

Finally, the configuration wizard will display a summary of the tasks to be performed for you to review before continuing. Once the wizard finishes, you'll be provided with a summary of the steps taken and their results, as shown in Figure 21-12.

Service configuration

Once the wizard finishes, you can return to the ADFS management console. The first thing you'll want to do is right-click the *Service* folder and click Edit Federation Server Properties. Here you can configure key details about your ADFS environment that will be published to parties you're federating with.

On the General tab, you can configure a display name for your service. On the Organization tab, you can publish contact information for your ADFS support team that will be listed in the ADFS server's federation metadata. Federation metadata is used to automatically configure the bulk of a trust relationship between parties. Finally, on the Events tab, you can adjust the logging and auditing for the service.

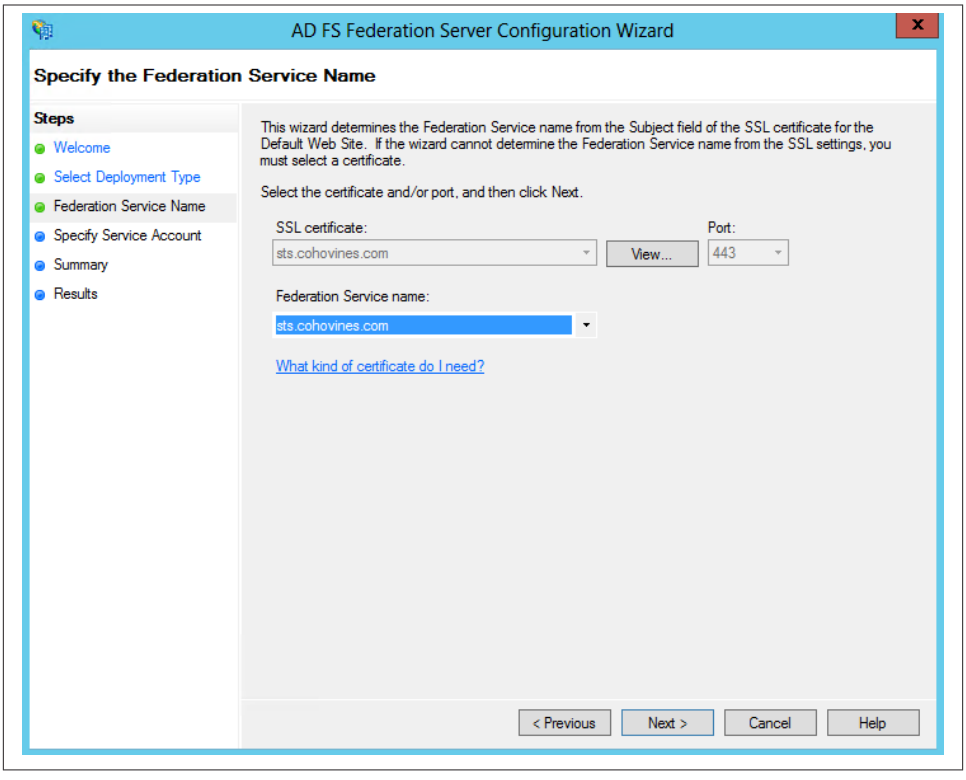
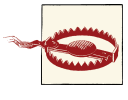


Figure 21-11. Federation service name configuration



The "Federation Service name" and "Federation Service identifier" values on the General tab must be set during initial configuration. If you change these values after you have deployed ADFS, you may break the trust relationships you have with other parties!

The second configuration item that you should address before you begin federating is the topic of token signing and token encryption certificates. As we discussed earlier in the chapter, every token ADFS issues is signed with the ADFS token signing certificate. The organizations that you trust will keep a copy of the ADFS server's public key locally so that they can validate each token they receive from your ADFS environment.

The problems that arise here are twofold. First, the token signing and token encryption certificates are self-signed. Self-signed certificates are generated locally on a server and are not issued by a trusted CA. Some federation platforms do not consider tokens signed by a nontrusted (e.g., self-signed) certificate to be valid, even if the signature is cryptographically valid. In these cases, you must either convince the other party to trust your token signing certificate, or use a token signing certificate from a commercial CA.

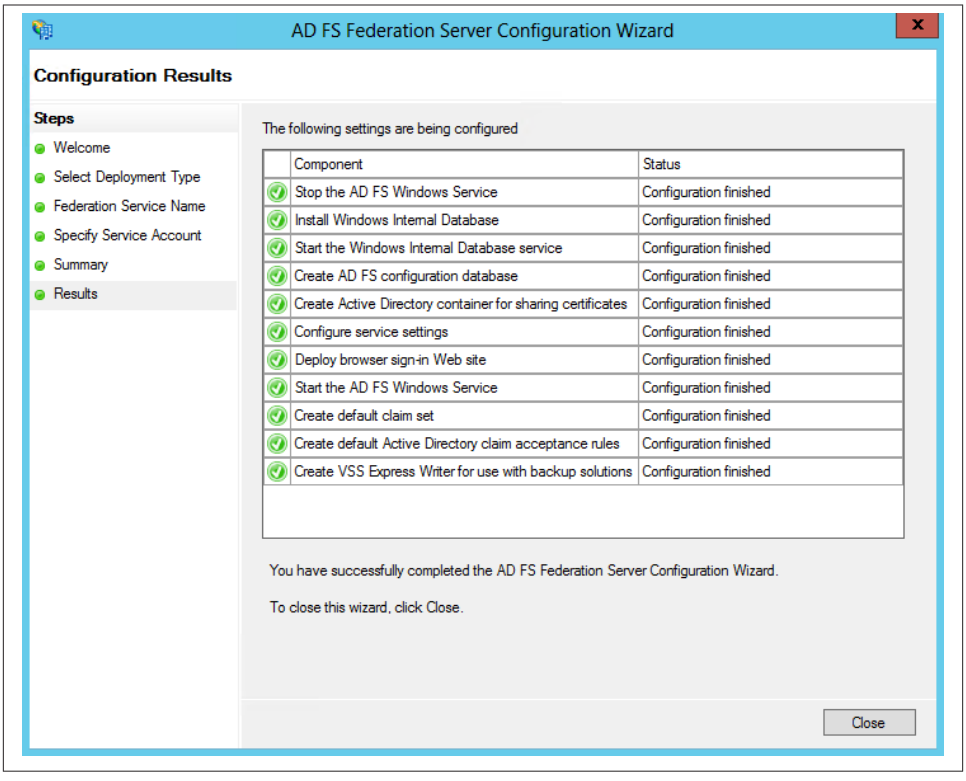


Figure 21-12. Wizard results summary

The second issue is the topic of certificate expiration. Since the parties you have trusts with must keep a copy of your token signing certificate's public key on hand, they also need a mechanism to update that key when it expires. There is a protocol in place for relying parties to automatically pick up new token signing certificate public keys when they become available, but many federation platforms (and applications) don't implement this. Consequently, when the certificate expires, the trust relationship will fail and access to the federated application will be impacted.

Since the certificate used for token signing is so important and difficult to change later, we recommend that you obtain a commercially issued token signing certificate that expires as far in the future as possible. It is well worth the added expense of buying a certificate with three (or more) years of validity up front, when you consider the cost of rotating the certificate after a year or two. The subject name on the token signing certificate does not matter at all. We typically append "-signing" to the FQDN used for the service certificate (e.g., *sts-signing.cohovines.com*). In addition, the CA that you purchase the certificate from must include the Certification Authority Issuer OID in the Authority Information Access (AIA) field of the certificate (see Figure 21-13 for an

example). This is an obscure requirement that you'll need to research with your CA before purchasing the certificate. At the time of publication, we know that DigiCert (mentioned earlier) and Entrust include these fields in their certificates.

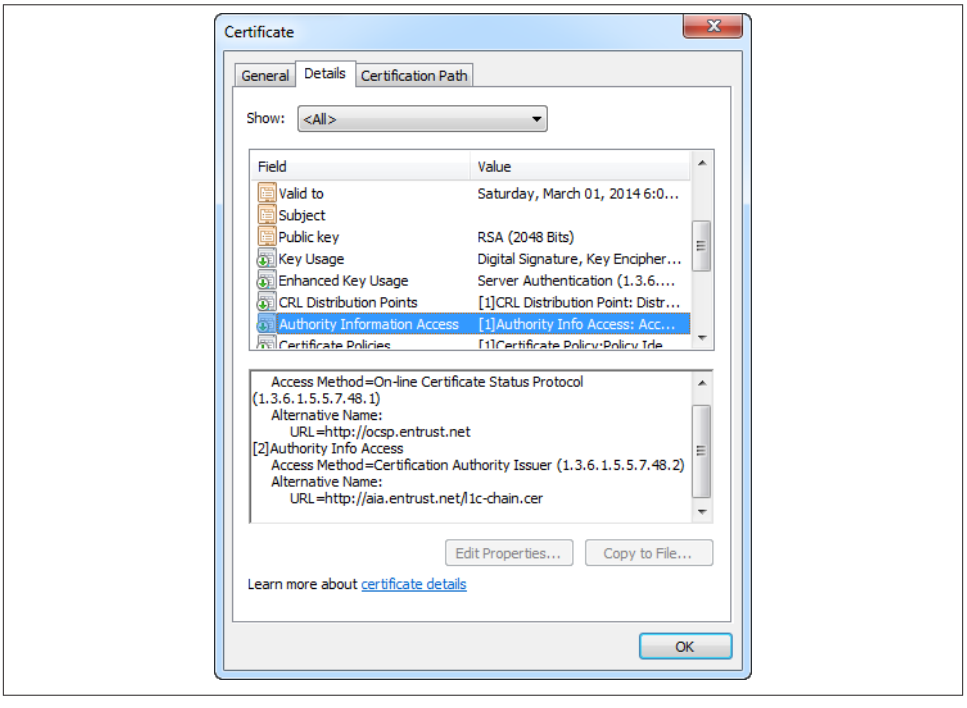


Figure 21-13. Certificate details

We've mentioned the token encryption certificate but haven't discussed it thus far. The reason for not discussing it is simple—we don't recommend that you enable token encryption in any of your trust relationships. The contents of any transaction (including the token) between ADFS and an end user or federated party are already protected via SSL in the browser. When you enable token encryption, the token itself is encrypted using the token encryption certificate. If you need to troubleshoot the interaction between ADFS and another party, token encryption will make it nearly impossible to do so.

Federation Server Proxies

Deploying the federation server proxy is extremely easy. Outside of the deployment wizard, it has no real settings to configure. The only steps involved are installing the ADFS server role and choosing "Federation Service Proxy" in the first step (Figure 21-14), and then running the wizard.

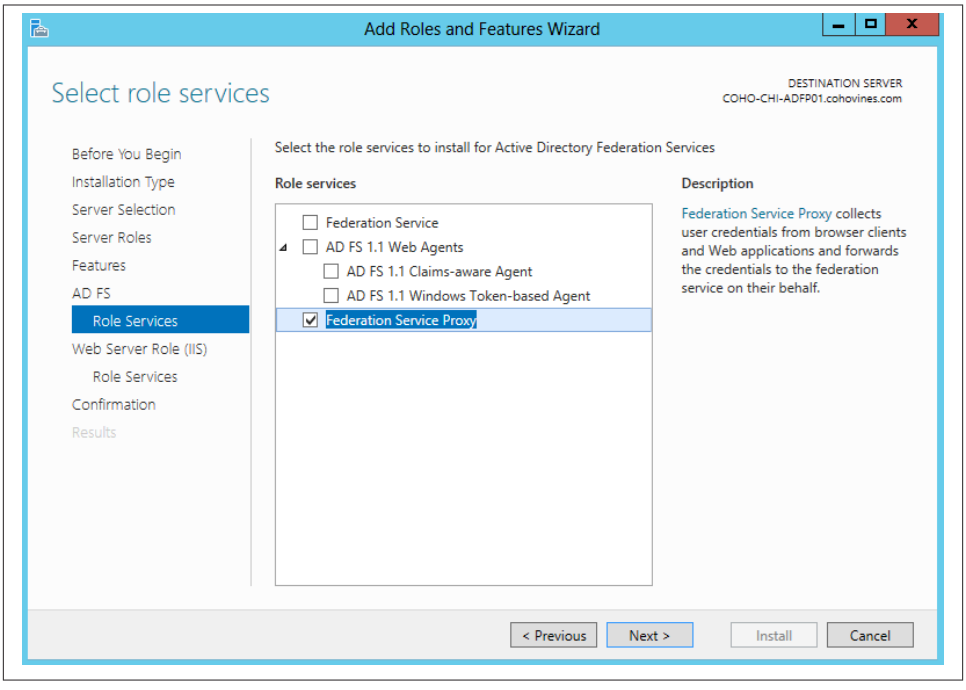


Figure 21-14. Add Roles and Features Wizard

As for the federation server role, you'll need to import a certificate for use with the IIS website that ADFS uses. You can use the DigiCert utility discussed earlier to do this. In fact, you can simply import a copy of the certificate in use on your federation servers: just use the DigiCert utility to export that certificate to a PFX file and then import it on the federation server proxy.

Once you have imported the certificate, launch the Internet Information Services Manager MMC snap-in. Browse to *Sites\Default Web Site* in the Connections pane on the left. Next, in the Action pane on the right, click Bindings, as shown in Figure 21-15. Click Add, change the Type to "https", and select your certificate from the SSL certificate drop-down, as shown in Figure 21-16.

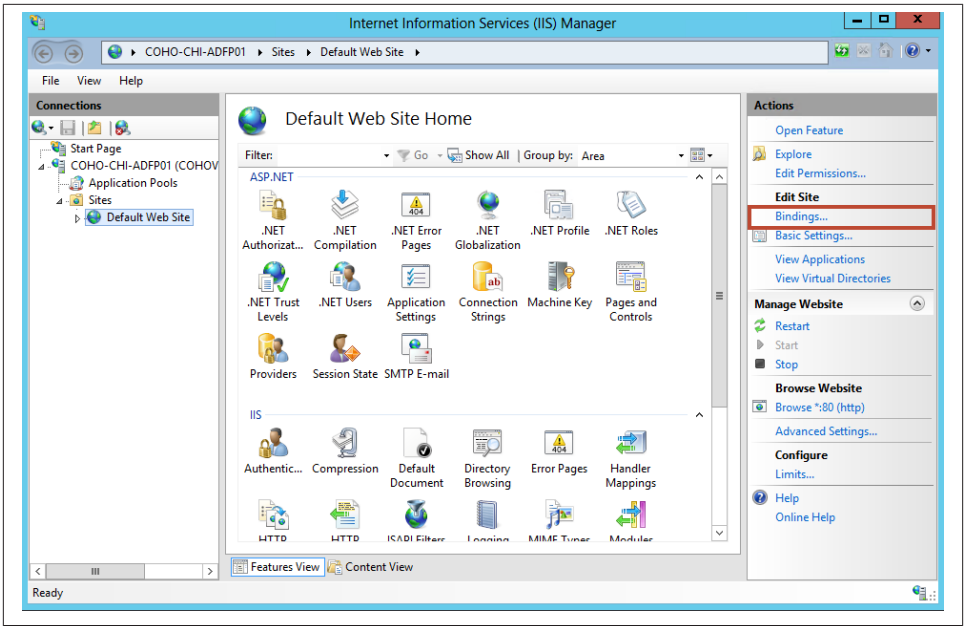


Figure 21-15. IIS Manager

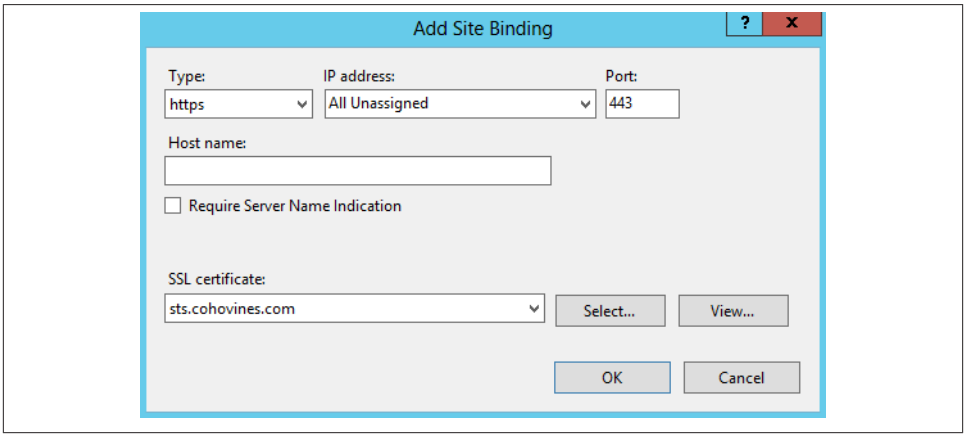
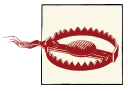


Figure 21-16. Add Site Binding dialog

Next, you must configure the hosts file on your federation server proxy to resolve the name of the federation service to your federation server or the load balancer VIP in front of your federation server(s). You can do this by opening the hosts file in *%windir%\system32\drivers\etc* with Notepad from an elevated command prompt.

At the bottom of the file, enter the IP address of your federation server or load balancer VIP, press Tab, and then enter the federation service name (e.g., *sts.cohovines.com*) on a new line, as shown in Figure 21-17.



It is very important that you separate the IP address and hostname with a tab!

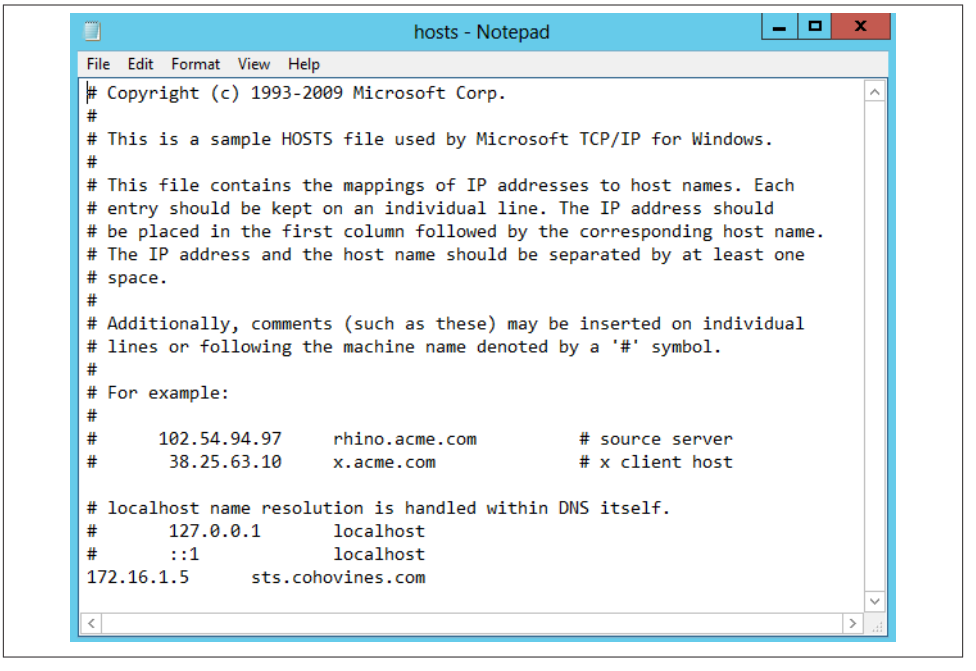


Figure 21-17. Editing the hosts file

Once you have completed editing the *hosts* file, launch the AD FS Federation Server Proxy Configuration Wizard. The wizard will automatically populate your federation service name based on the SSL certificate imported in IIS. Click the Test Connection button shown in Figure 21-18 to ensure that there are no connectivity issues (such as firewalls or name resolution problems) before continuing.

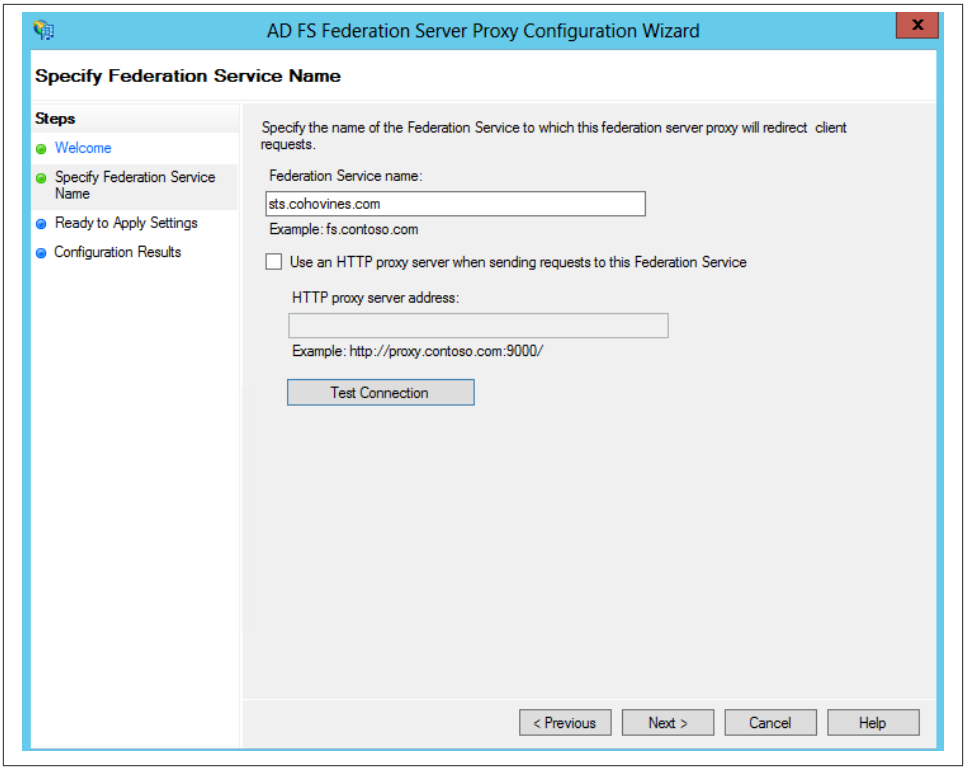


Figure 21-18. The AD FS Federation Server Proxy Configuration Wizard

When you click Next, you will be prompted for administrative credentials to the federation server farm. These are only used to establish the relationship between the proxy and the federation servers. The credentials you enter must have local administrator access to the federation servers. If the credentials are valid, you will be presented with a summary of the steps that will be taken.

Figure 21-19 shows the results of a successful federation proxy server configuration. There are no further steps necessary to complete the initial configuration.

Relying Party Trusts

As a typical corporate user, most of the configuration work you'll be doing in ADFS will most likely be in the realm of relying parties. The term *relying party* (RP) is ADFS's nomenclature for what you would typically think of as an application that relies on your ADFS infrastructure for authentication. This could be an internally developed application that supports claims, a cloud-hosted service such as *salesforce.com*, or an outsourced human resources and payroll application, for example.

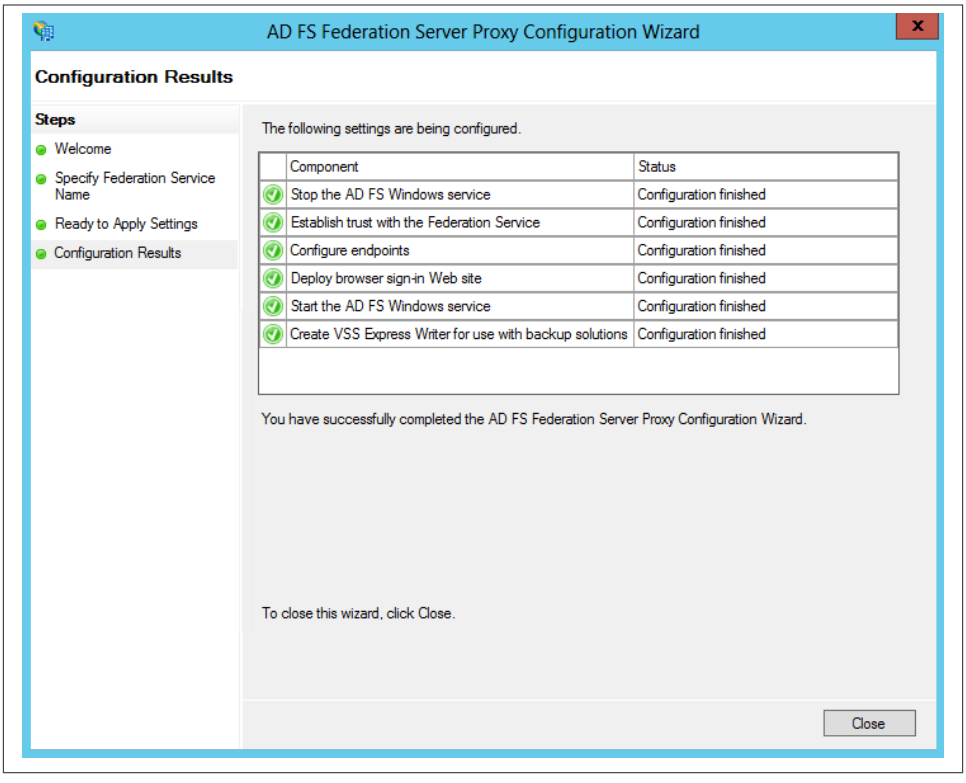


Figure 21-19. Federation proxy configuration results

There are a couple of scenarios that come into play when you're ready to set up an RP. In the best case, the RP provides a *federation metadata document*. This is an XML document published by the RP on its server (or sometimes provided to you directly as a file) that contains all of the necessary details to configure the trust relationship. If you want to see the federation metadata for your ADFS environment, browse to this link, replacing the FQDN with the relevant value for your environment.

If your RP doesn't provide federation metadata, you'll need to enter a number of values manually in the Add Relying Party Trust Wizard. For the sake of completeness, we'll walk through the scenario with an application that doesn't publish federation metadata.

To get started, open the ADFS management console and browse to *AD FS\Trust Relationships\Relying Party Trusts*. Right-click and then click Add Relying Party Trust. Start the wizard and choose "Enter data about the relying party manually," as shown in Figure 21-20.

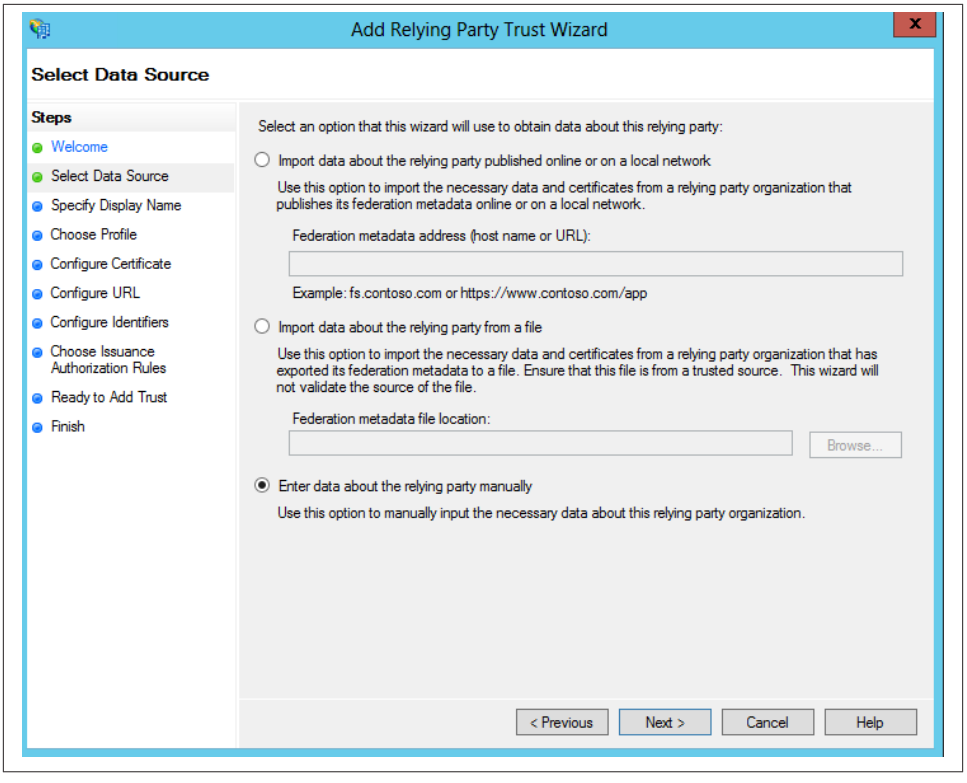


Figure 21-20. Adding relying party trust manually

Next, provide a name for the application and a description. In this example, we'll call the application "Federation Demo." We recommend that you use the description field to track relevant information about the application, such as the owner of the relationship with that application in your organization and the date the trust was established.

On the Choose Profile screen, use the default of ADFS profile. The Configure Certificate screen allows you to specify a token encryption certificate to use. You will need to import the public key of the relying party's token encryption certificate to manually configure it here. As discussed earlier in this chapter, we don't recommend the use of token encryption certificates in order to ease troubleshooting.

Figure 21-21 shows the Configure URL screen. This is where you will specify the URL that ADFS should redirect the user to once ADFS has issued a token. You must specify whether the application supports WS-Federation or SAML 2.0 on this screen. In this case, our application supports WS-Federation and the user should be redirected to this link. You will need to collect this URL from the RP's support team before you begin the trust setup process.

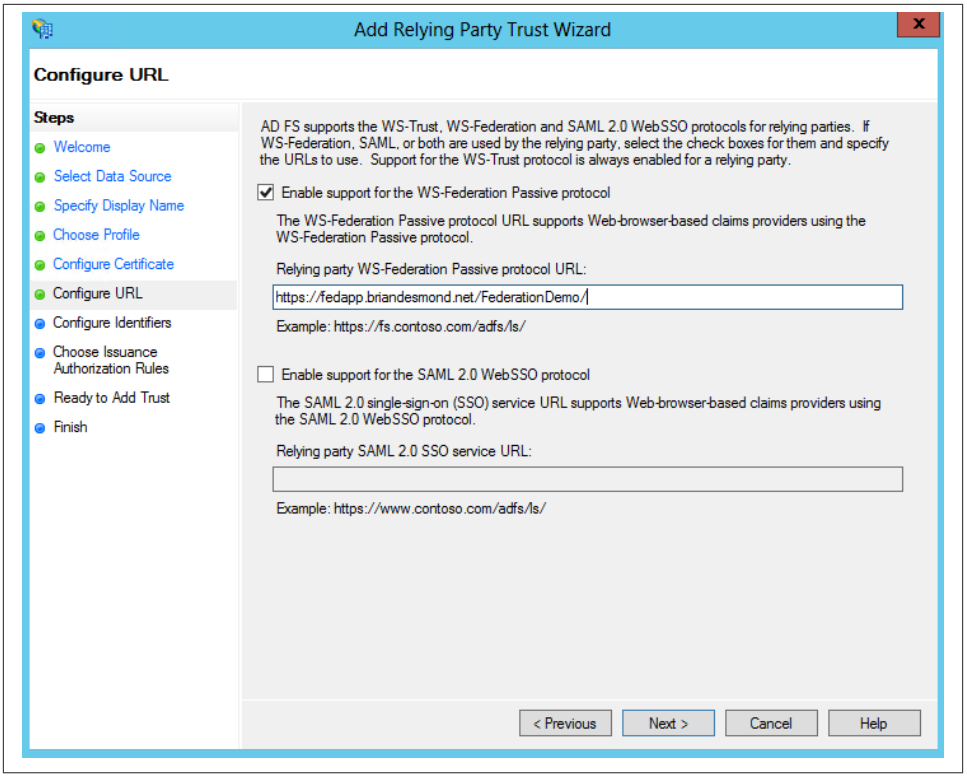


Figure 21-21. Configuring the relying party trust URL

Next, you will need to specify the identifier(s) that the RP provides. This is how ADFS identifies a request from a given application. By default, the URL you provided in Figure 21-21 is populated here, as shown in Figure 21-22. The value can be arbitrary, essentially, and it is included in the authentication request from the application. Like the URL, you will need to gather this data from the RP's support team in advance.

Click Next, and elect to simply "Permit all users access to this relying party" on the Choose Issuance Authorization Rules screen. We'll discuss issuance authorization rules briefly in the next section on claims rules. The confirmation screen shown next will provide you with a tabbed summary of all the choices you made in the wizard. All of these choices can be changed later, but you should take a moment to review them and make sure there are no obvious mistakes.

By default, the final screen of the wizard will allow you to launch the claims rules editor for the new RP trust once you finish. We will look at claims rules in more detail in the next section; in a nutshell, they allow you to define what claims should be sent to the RP and whether or not those claims should first be transformed before passing them on to the RP.

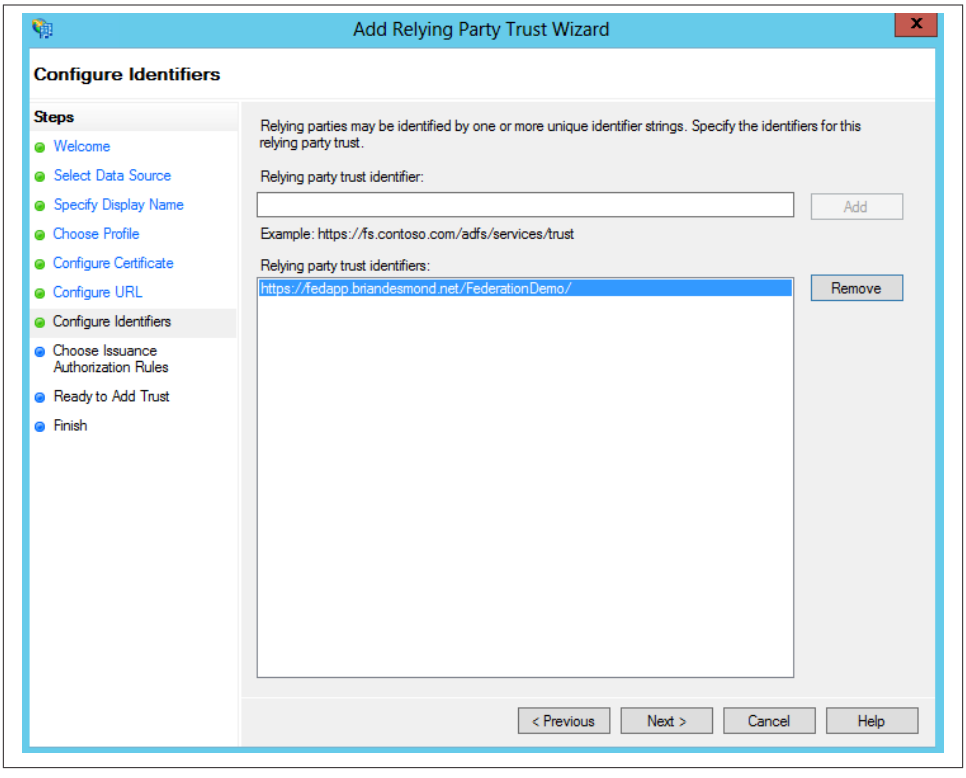


Figure 21-22. Configuring the relying party trust identifier

Claims Rules and the Claims Pipeline

In the previous section, we created a relying party trust with a sample application. At the end of the process we had an established trust, but we didn't configure ADFS to send any claims to the application. Without sending any claims, the value of the RP trust is quite limited.

In this section, we'll talk about two extremely important concepts: the claims pipeline and claims rules. The claims pipeline is the process through which every claim that ADFS processes passes. Claims rules are used to inject and filter claims in the pipeline as they make their way onward to a relying party.

The Pipeline

There are a number of stops that claims pass through on their way to a relying party. For the purpose of this discussion, we'll assume you are using Active Directory as your sole claims provider. Figure 21-23 shows a diagram of the claims pipeline.

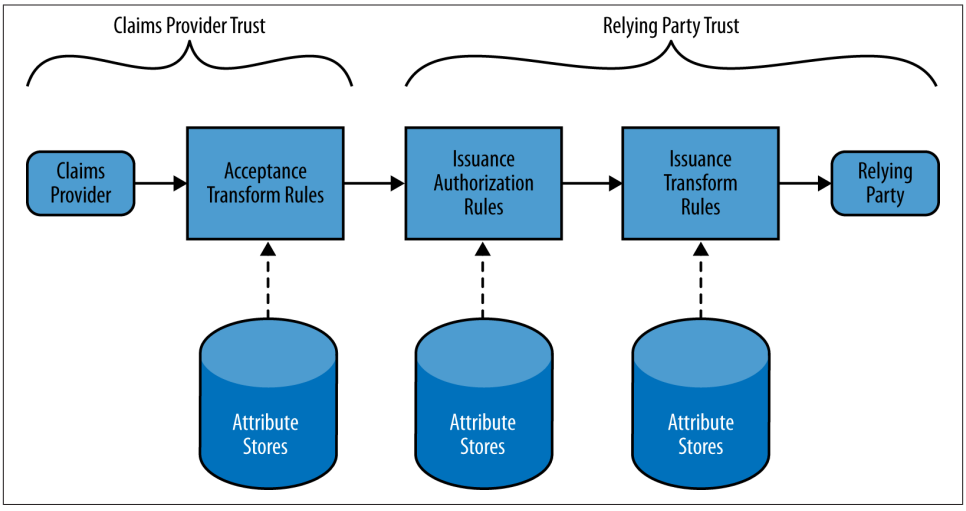


Figure 21-23. The claims pipeline

The claims provider is the where claims enter the pipeline. Claims providers provide claims to ADFS. You use acceptance transform rules to determine what claims from the claims provider are available for consumption by relying party trusts. If you look at the default acceptance transform rules for the Active Directory claims provider trust (Figure 21-24), you'll notice that the list is quite bare.

Perhaps a bit counterintuitively, all of the attributes of a user aren't available from the AD claims provider—just some special attributes such as group memberships, SIDs, and authentication info. Specific attributes must be requested from the Active Directory attribute store. We'll discuss this further shortly.

Once claims have passed through the claims provider's acceptance transform rules, they are presented to the relying party trust in question. The first step is to make sure that the user is permitted to authenticate to the given relying party. This authorization step happens in the RP's issuance authorization rules. By default all users are authorized to use an RP, as evidenced by the "Permit Access to All Users" rule shown in Figure 21-25. You can access any of the claims from the claims provider trust, or inject more claims into the pipeline from an attribute store in order to make an authorization decision. ADFS determines whether or not a user is authorized based on the presence of a claim called Permit with a value of `true` in the pipeline.

Finally, the claims set can be further amended or transformed using the RP trust's issuance transform rules. The claims that are present at the conclusion of the processing of the issuance transform rules are the claims that are sent to the RP. By default, an RP trust has no issuance transform rules. This effectively means that the RP will receive no claims about the user. Let's look at how to change this.

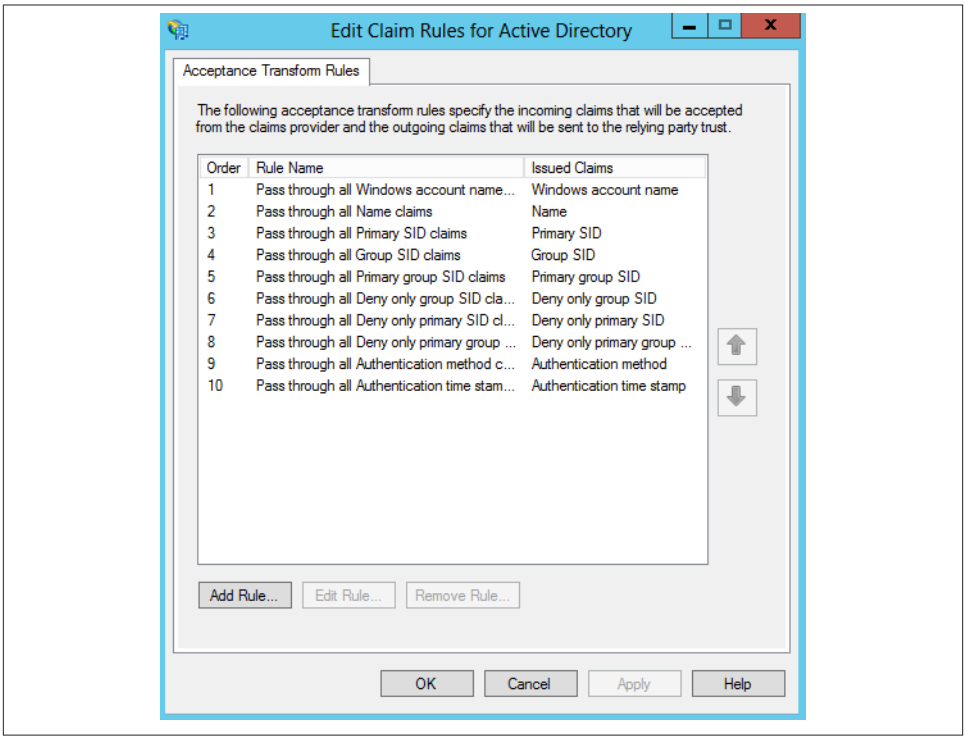


Figure 21-24. Acceptance transform rules for Active Directory claims provider trust

Creating and Sending Claims Through the Pipeline

There are two key ways you can generate claims to be sent to an RP. The first is via the claims provider trust, and the second is via the RP trust's issuance transform rules. The advantage of defining your claims set on the claims provider trust is that you can have a central set of claims that are available to all RP trusts. This can reduce the management burden and limit the number of places you need to look when troubleshooting or making global changes.

To edit the claims rules for the Active Directory claims provider trust, browse to *AD FS \Trust Relationships\Claims Provider Trusts*. Select Active Directory and then click Edit Claim Rules in the task pane on the right. In order to add additional claims in the form of user attributes to the pipeline, click Add Rule.

There are five claims rule templates you can choose from. Here is a brief description of each of these templates:

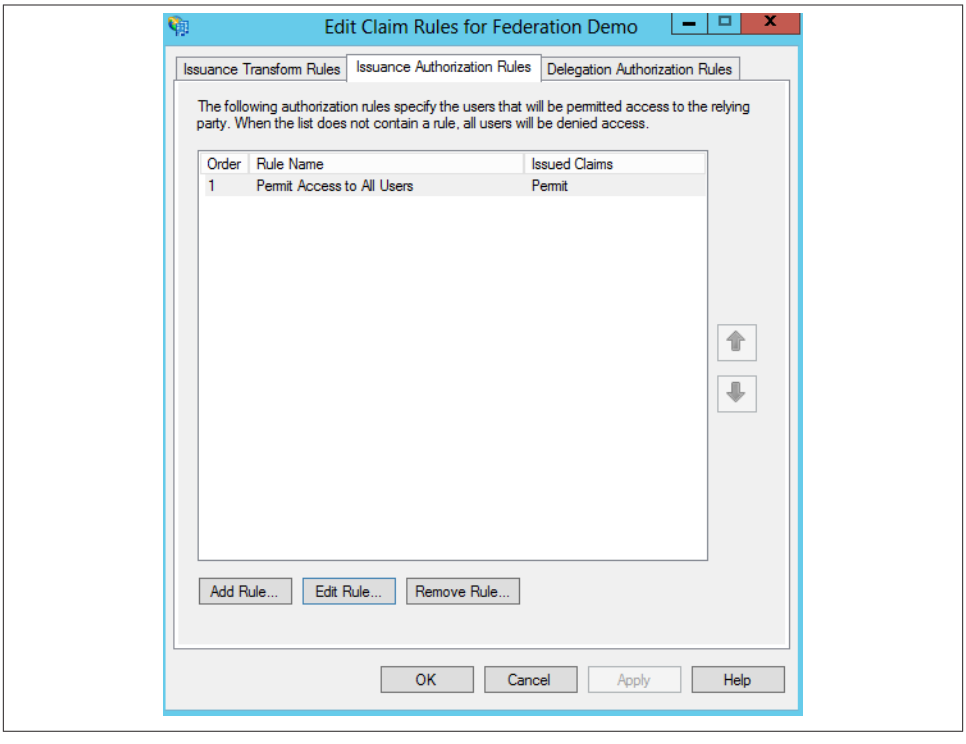


Figure 21-25. Issuance authorization rules

Send LDAP Attributes as Claims
: This rule will inject one or more attributes of the user into the claims pipeline. In addition to injecting Active Directory attributes, you can use other LDAP-based attribute stores such as an AD LDS server.

Send Group Membership as a Claim
: If you need to insert a claim into the pipeline based on whether or not a user is in a group, use this rule. For example, if you wanted to have a claim called IsSupervisor and all of the supervisors in your organization were in a group called Supervisors, you could use this rule.

Transform an Incoming Claim
: This rule enables you to perform very basic transformations on claims that are already in the pipeline. You can change the type of a claim, perform basic string replacements, or transform an email suffix.

  As you consider this rule template, notice in Figure 21-24 that each claims rule is numbered. Rules are processed sequentially. In other words, each rule has access to

all of the claims that have been added to the pipeline before it within the ordered list (and earlier in the pipeline overall).

Pass Through or Filter an Incoming Claim
: This rule is similar in nature to the previous template. You'll primarily use this template on relying party trusts. You use the pass-through function to instruct ADFS to emit a claim in the pipeline to the RP. The filtering function is useful for emitting claims only if they match a specific value or if they start with a specific value. If the claim doesn't match the filter rule, it won't be passed through.

Send Claims Using a Custom Rule
: ADFS has an entire custom language called the *claims rule language*. This language is used for expressing complex custom expressions that determine whether or not to emit a claim, enable you to perform complex transformations on claims values, and enable access to custom attribute stores and SQL attribute stores.

  Explaining the claims rule language is a substantial topic and not one that we have room for in this chapter. Microsoft has documented many common examples online, and we encourage you to start there if you want to take on custom claims rules.

Now that we've discussed the types of rule templates available, let's make use of them to add a few claims to the pipeline. In the wizard, select the "Send LDAP Attributes as Claims" rule template. Create a rule to send the user's name and location attributes, as shown in Figure 21-26.

In Figure 21-26, we are sending the user's first and last name, display name, and city and country attributes as claims. You probably will have noticed by now that the default options in the drop-down lists are missing most attributes. You can simply type in an Active Directory attribute name if you don't see what you want. If you need to add additional outgoing claim types, you'll need to create claim descriptions for those attributes. Refer to the sidebar "Creating Claim Descriptions" for more information on how to do this.

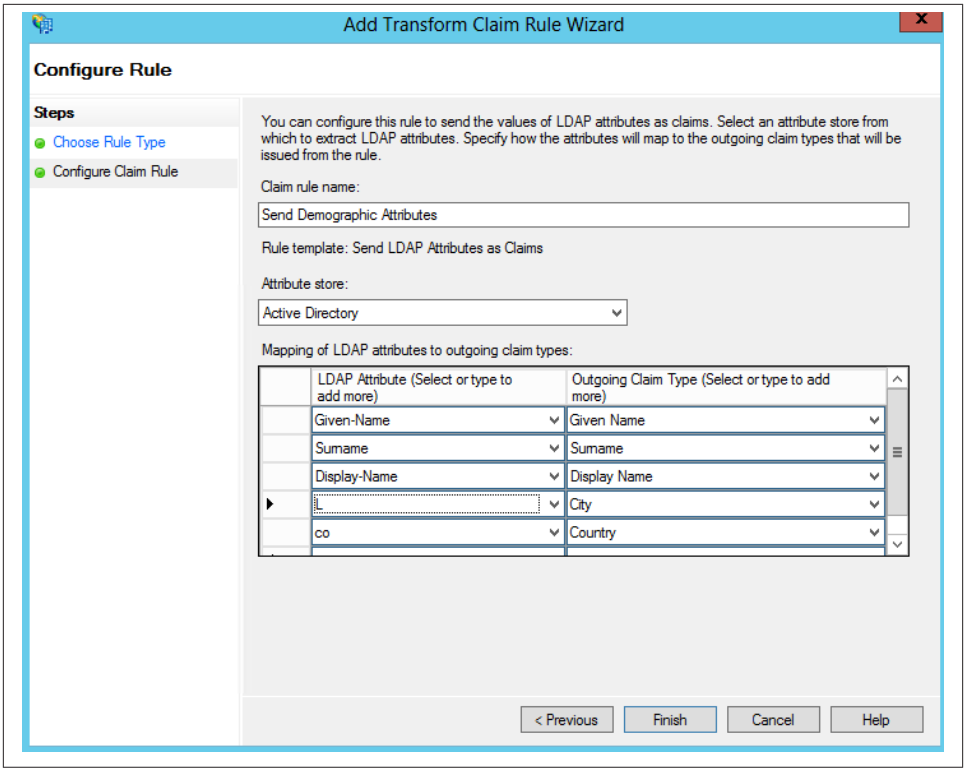


Figure 21-26. Send LDAP attributes as claims

Creating Claim Descriptions

Claim descriptions provide a unique identifier for a claim in the form of a uniform resource identifier (URI), as well as flags that control whether or not the claim is published in the ADFS environment's federation metadata. Publication of claims in the federation metadata is entirely optional; it provides a way for you to tell your federation partners which claims you accept (in the case of a claims provider trust) and which claims you publish (in the case of relying party trusts).

To create a new claim description, browse to *AD FS\Service\Claim Descriptions* in the ADFS management console and click Add Claim Description in the task pane on the right.

Populate the dialog shown in Figure 21-26 as appropriate for your claim. The claim identifier should be a well-formatted URI, but it does not need to actually point to anything. The URI is simply the unique identifier for the claim. You can optionally check the bottom two boxes if you want to publish metadata for this claim.

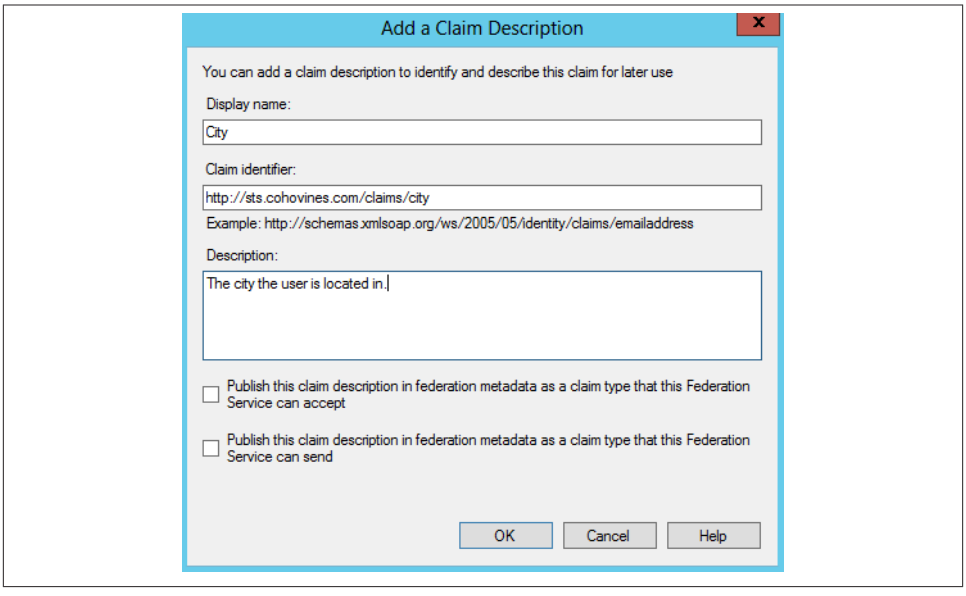


Figure 21-27. Add claim description dialog

When you've completed the wizard, click Add Rule again. This time, we'll use the "Send Group Membership as a Claim" template to create a claim for full-time employees. Figure 21-28 shows how you can specify a group and then set the value of a claim if the user is a member of that group.

Now that we've added some claims that will be available to all of our RP trusts, let's configure the RP trust we created earlier to emit these claims. Find the Federation Demo RP trust in the ADFS management console, and click Edit Claim Rules. On the Issuance Transform Rules tab, click Add Rule.

First, note that the templates that were available earlier when you were editing the claims provider trust claim rules are also available here. If you want, you could perform the same configuration steps we just executed on the RP trust instead. This would have the effect of only making those claims available to this RP. Since we already have the claims we need in the pipeline, we'll simply pass them through.

Use the "Pass Through or Filter an Incoming Claim" template and configure it to pass through the Given Name claim, as shown in Figure 21-29. Repeat this for each of the other claim types we configured:

- Surname
- Display Name
- City

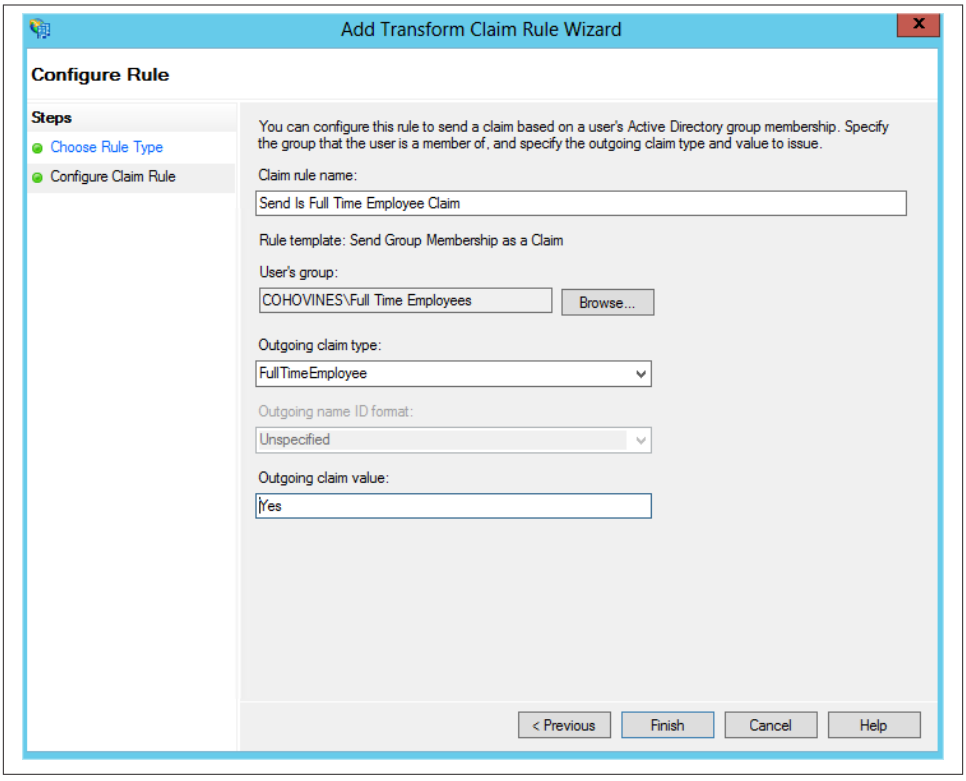


Figure 21-28. Send a claim based on group membership

- Country
- FullTimeEmployee

Unless you have customized your ADFS implementation already, you'll need to complete the steps in the sidebar "Creating Claim Descriptions" on page 642 to add the Display Name, City, Country, and FullTimeEmployee claims to ADFS.

Once you're done, your issuance transform rules should look similar to Figure 21-30.

Figure 21-31 shows the results of the claims rules we just created. The screenshot shows the output from the sample application available at this link. This application simply outputs all of the claims the federation server sends. Once configured, this is a useful tool for testing your ADFS claims rules.

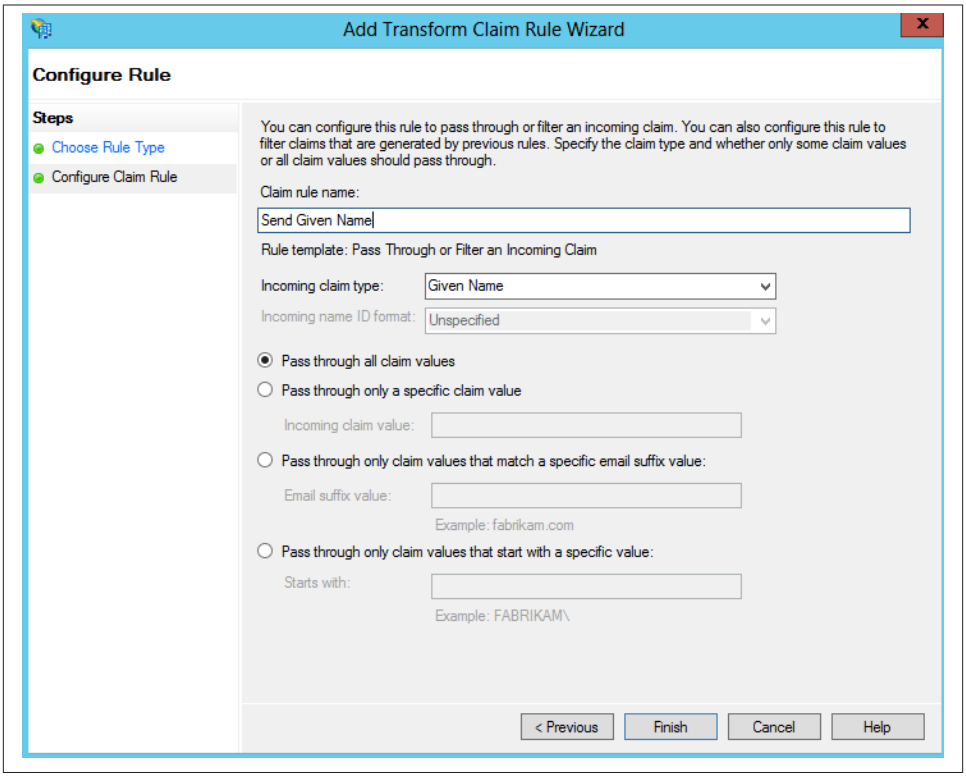


Figure 21-29. Passing through the Given Name claim

To summarize, in this section we configured our claims provider trust (Active Directory) to add a number of claims to the pipeline for consumption by all of the relying parties in the system. To do this, we first added claim descriptions for each custom claim, and then we used the "Send LDAP Attributes as Claims" template to add attributes of the user to the pipeline and the "Send Group Membership as a Claim" template to add a claim if the user is a member of a particular group. Finally, we added pass-through rules to the issuance transform rules of the RP trust to send the claims to the RP.

Customizing ADFS

With a bit of configuration, ADFS is usable out of the box. Chances are, though, you'll want to customize the experience a bit. Depending on where your goals for customization lie, the scope of this project may vary quite a bit. The two main customizations that organizations take on are look/feel (or functionality) adjustments for the forms-based logon and home realm discovery pages, and the addition of custom attribute stores.

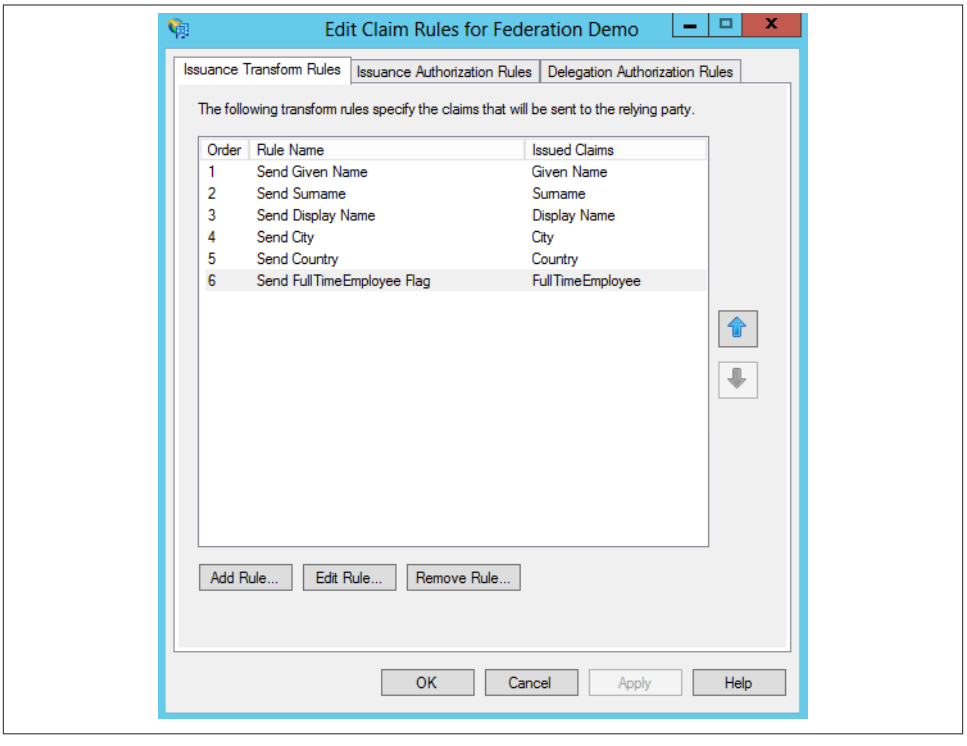


Figure 21-30. Issuance transform rules for RP trust

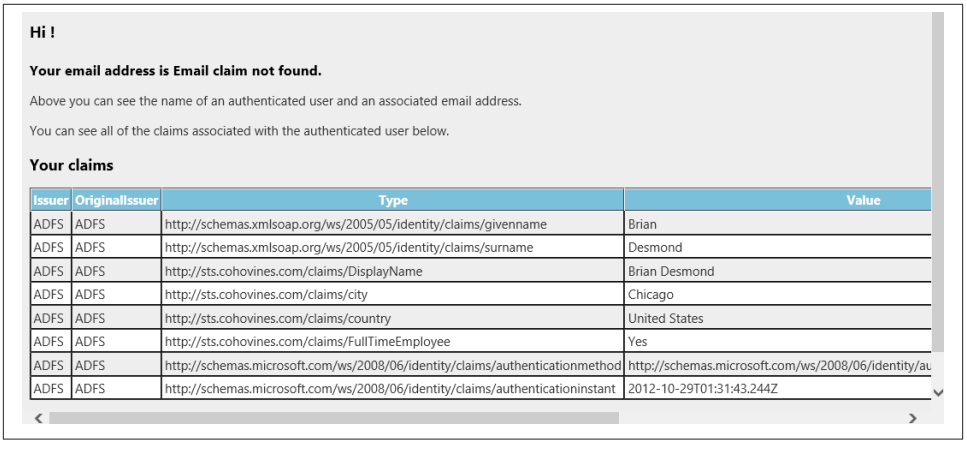


Figure 21-31. Claims sent to demo application

Forms-Based Logon Pages

When accessing ADFS from outside the corporate network through an ADFS proxy, your users will, by default, log on using the forms-based logon pages. The default experience for these pages is bland and gray. You'll probably want to work with your Web or Marketing Department to customize these pages to match your corporate brand and be more visually appealing.

Microsoft has a guide for customizing the pages available at microsoft.com. The level of complexity of the modifications you want to make (e.g., changing colors or adding a logo) will dictate whether or not this is a task best given to an ASP.NET developer. All of the files you can customize are located in *c:\inetpub\adfs\ls* on your ADFS servers. Be sure to make a backup copy of the original pages before you begin in case you need to roll back.

If you'd simply like to add a logo, edit the *web.config* file in *c:\inetpub\adfs\ls* and change the `logo` setting to the name of a logo file (or just save that file as *logo.png*). Microsoft recommends that your file be 600 pixels wide and 100 pixels tall for best results.

Attribute Stores

Attribute stores are used to provide claims from external systems. Out of the box, ADFS supports sourcing attributes from Active Directory, LDAP servers such as AD LDS, and SQL servers. If you need to provide claims from other systems (e.g., an HR system or an Oracle database), you'll need to develop a custom attribute store to do that.

Developing a custom attribute store requires .NET development skills and is outside the scope of this book. Documentation for getting started is available at microsoft.com. Be sure to test your implementation thoroughly, especially with regard to error handling and performance.

Troubleshooting ADFS

Troubleshooting ADFS will likely come naturally to web developers, as some of the key tools used are traditional web debugging tools. For an IT pro, the learning curve will be much higher, as the amount of data you can get from traditional event logs is relatively limited.

Before we look at a couple of tools in depth, the first trick to have in your toolbox is Internet Explorer's (IE) InPrivate browsing mode. InPrivate prevents access to or saving of cookies and masks your browsing history, improving security. You can turn on InPrivate mode in IE by pressing Ctrl-Shift-P. Alternatively, you can right-click the Internet Explorer icon in your taskbar and click Start InPrivate Browsing. When you're in InPrivate mode, you'll see an InPrivate identifier to the left of the address bar as well as a help menu on the startup page, as shown in Figure 21-32.

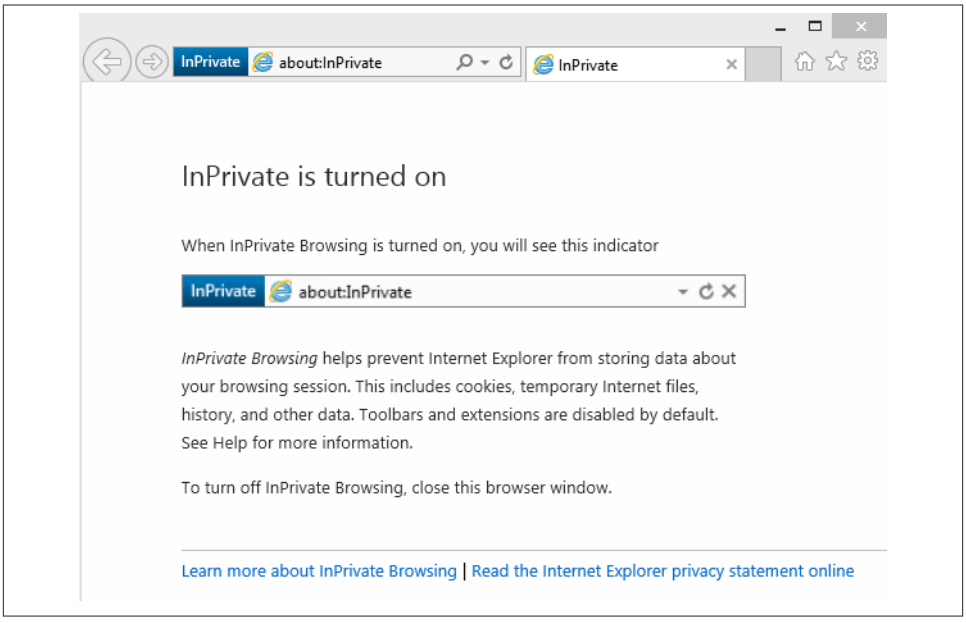


Figure 21-32. Internet Explorer InPrivate Mode

Event Logs

ADFS includes a number of event logs as well as a useful data point called a *correlation identifier* to assist you in troubleshooting issues. Any time an issue is encountered by a user in the ADFS web application, an error screen similar to Figure 21-33 will pop up. Of particular interest to you as the administrator is the reference number field.

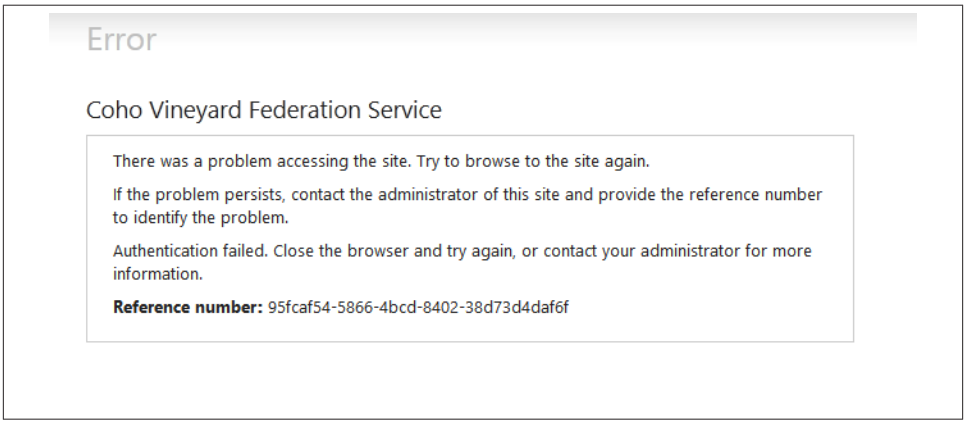


Figure 21-33. Error page with reference number

To review the ADFS logs and correlate them with the reference number shown in Figure 21-33, launch the Event Viewer on your federation server and browse to *Applications and Services Logs\AD FS\Admin*. Next, go to View→Add/Remove Columns. Add the Correlation Id column and reorganize your displayed columns, as shown in Figure 21-34.

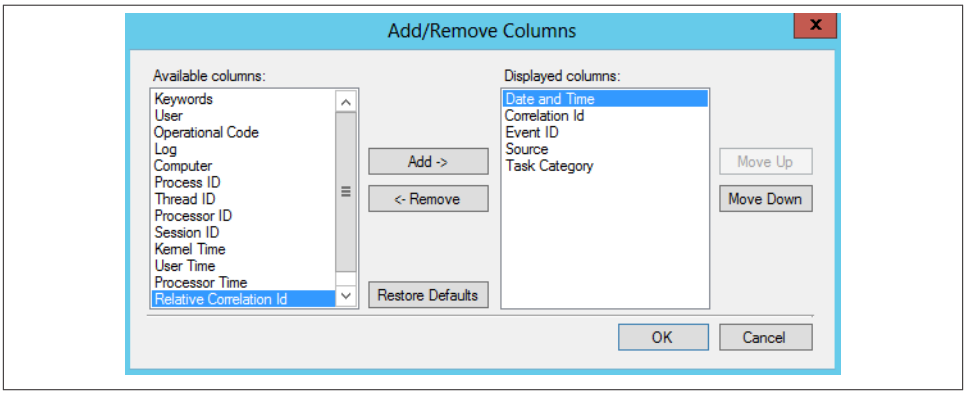


Figure 21-34. Add/remove columns from Event Viewer

Note that the reference number shown in Figure 21-33 matches the value in the Correlation Id column for two events. Specifically, the event in Figure 21-35 tells us that there is not a relying party trust configured with the identifier that was sent by the application.

If the data in the Admin event log isn't sufficient, you'll need to enable the trace log for ADFS. To do this, go to View→Show Analytic and Debug Logs. Browse to *Applications and Services Logs\AD FS Tracing\Debug* and then right-click the Debug log and click Enable Log. Reproduce your error and refresh the view. Figure 21-36 shows a helpful event confirming the diagnosis in Figure 21-35. When you're done, be sure to right-click the Debug log and click Disable Log.

You will need to repeat the same procedure to add the Correlation Id column to this log's display.

Fiddler

The most important tool for troubleshooting ADFS is a free one called Fiddler. Fiddler is an HTTP proxy that allows you to look at the HTTP traffic in a friendly GUI. If you've used Wireshark or Netmon before for network tracing, you can think of Fiddler as an application-layer version of those tools. Get a copy of Fiddler and install it on your

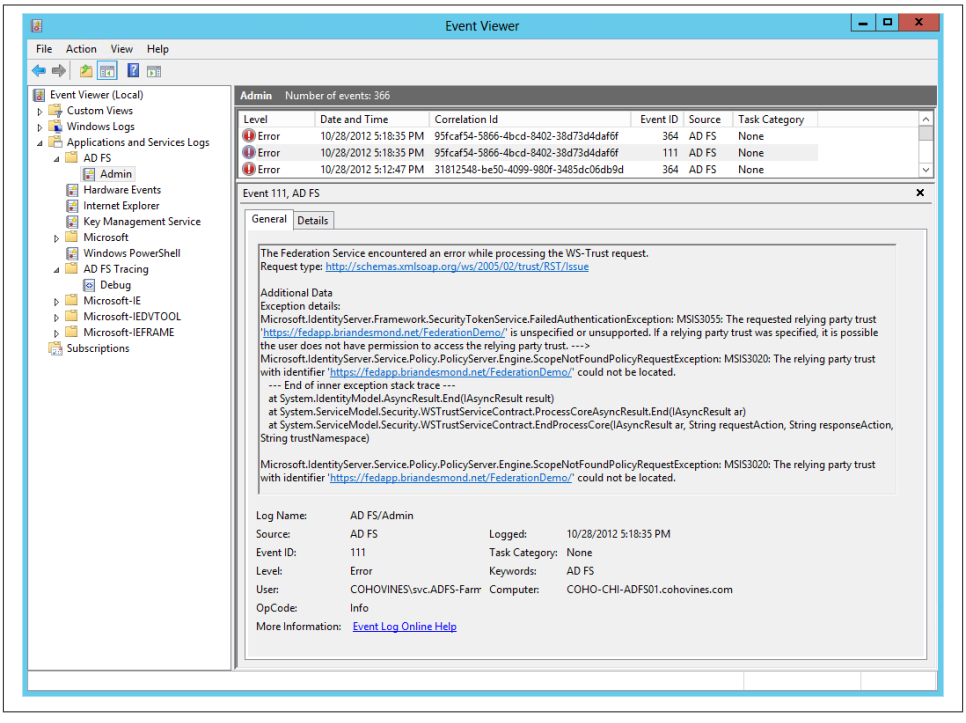


Figure 21-35. Error event with Correlation Id

workstation. Next, go to this link and install the add-ons for Fiddler that enable you to easily work with HTML, XML, and so forth.

Finally, download the Fiddler federation inspector from this site. This add-on will enable Fiddler to decode WS-Federation and SAML markup. Extract the *.zip* file and browse to the *bin\Debug* folder. Copy *Thinktecture.FederationInspector.dll* to *C:\Program Files (x86)\Fiddler2\Inspectors.* or wherever you installed Fiddler on your machine.

In order to look at encrypted ADFS traffic, you'll need to configure Fiddler to intercept HTTPS traffic. To do this, launch Fiddler and go to Tools→Fiddler Options. On the HTTPS tab, check "Capture HTTPS CONNECTs" and "Decrypt HTTPs traffic." Next, you will be prompted to generate a certificate for Fiddler to intercept SSL traffic and to configure Windows to trust the certificate, as shown in Figure 21-37. You'll receive another confirmation to import the certificate and then finally a final confirmation after a User Account Control (UAC) prompt. Answer Yes to all of these prompts.

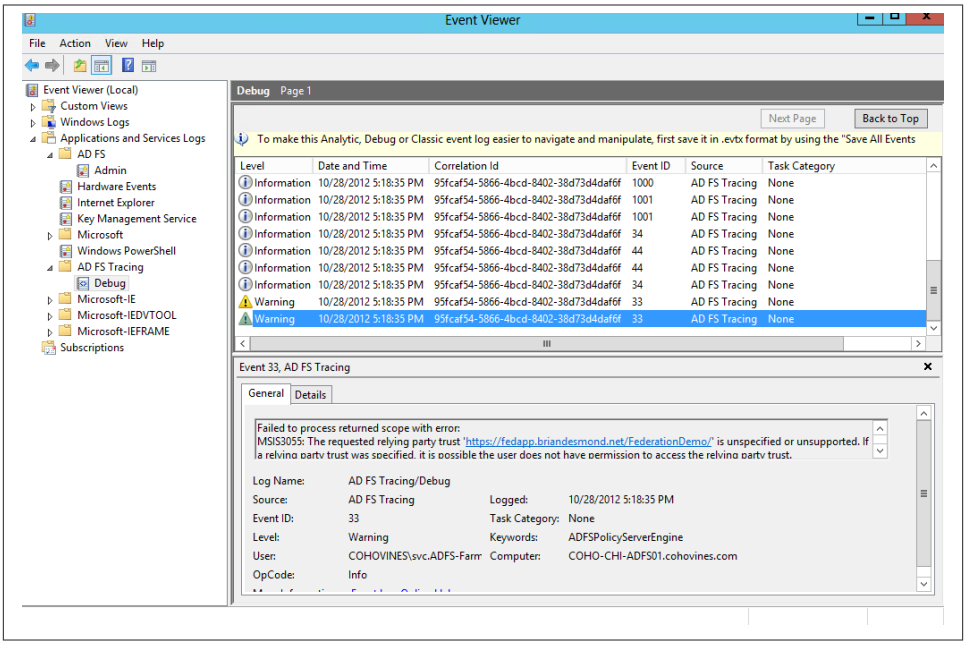


Figure 21-36. Tracing events in the Debug log

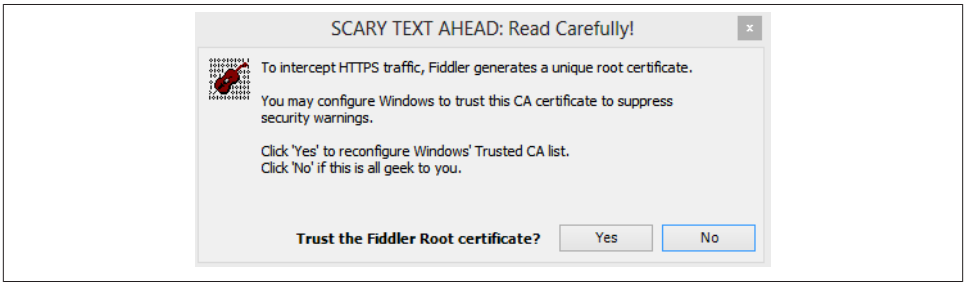


Figure 21-37. Configuring certificate trust in Fiddler

If you're using Windows Integrated Authentication on your ADFS server, you must disable Extended Protection before using Fiddler. See the sidebar "Configuring Extended Protection" on page 653 for details on how to do this.

Once you have completed these steps, browse to a federated application and log in. You'll see each of the steps in Fiddler that take place, as shown in Figure 21-38.

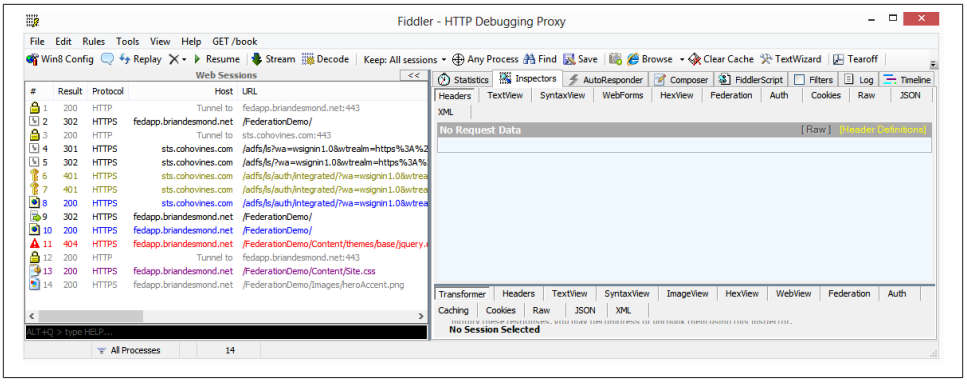


Figure 21-38. Fiddler trace of access to a federated application

Specifically, in Figure 21-38, the following steps happen (as shown in the "#" column):

2. The user browses to *https://fedapp.briandesmond.net/FederationDemo/*.
4. The application redirects the user to the ADFS server configured in the application.

The status codes 301 and 302 (Result column) are HTTP redirects.

5. The ADFS server performs an additional redirect to the correct authentication URL for the application.
6. The user is prompted to authenticate to the ADFS server using Windows Integrated Authentication. Figure 21-39 shows the Auth inspector in use in Fiddler that can be used to troubleshoot authentication.

The status code 401 notifies the browser that the user must authenticate to the application.

8. The user authenticates successfully to the ADFS server. Figure 21-40 shows the Federation inspector's decode of the WS-Federation token that was sent to the application.

The status code 200 signifies that the HTTP request completed successfully.

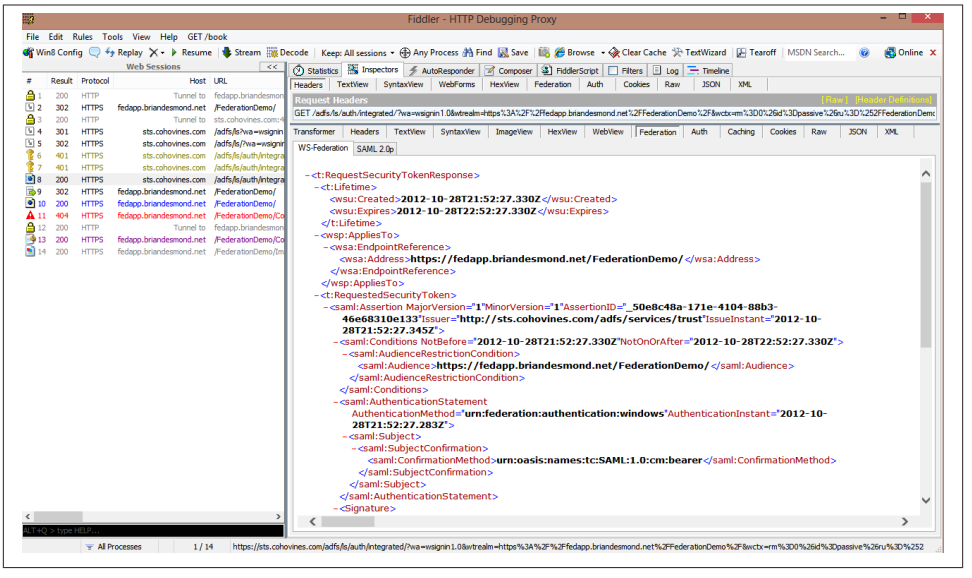


Figure 21-39. Fiddler Federation inspector

9. The ADFS server redirects the user back to the application based on the URL that was configured in the relying party trust in Figure 21-21.
10. The user successfully accesses the application.

Configuring Extended Protection

Extended Protection is a security feature that was introduced in a security update for Windows, as discussed at *http://support.microsoft.com/kb/968389*. In general, it is something you will want to have enabled; however, it directly impacts your ability to troubleshoot with Fiddler. If Extended Protection is enabled, you will be prompted continuously to authenticate by ADFS and will never be able to proceed to the application.

To disable Extended Protection for troubleshooting purposes, follow these steps on your federation server:

1. Launch the Internet Information Services (IIS) Manager. You can do this by going to Start→Run and entering **`inetmgr`**.
2. Browse to the *Sites\Default Web Site\adfs\ls* node in the Connections pane on the left.
3. Double-click the Authentication icon in the center pane.

4. Select Windows Authentication and then click Advanced Settings in the task pane on the right.
5. Change Extended Protection to Off, as shown in Figure 21-41.

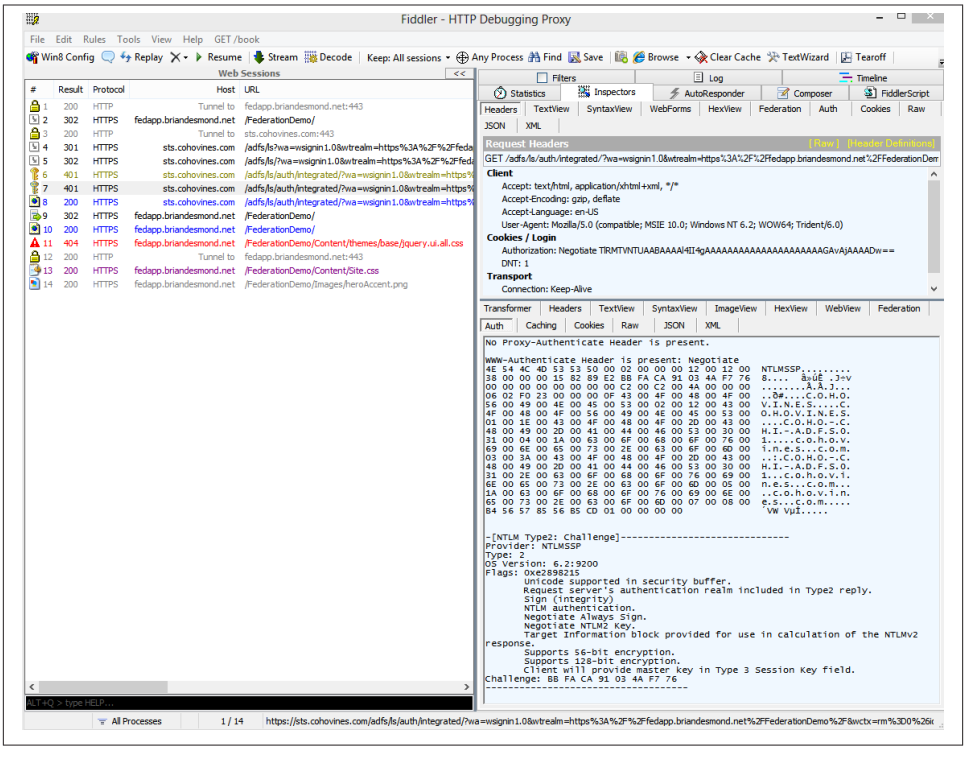


Figure 21-40. Fiddler Auth inspector

Summary

In this chapter, we looked at Active Directory Federation Services (ADFS). ADFS is Microsoft's answer to a crowded space full of vendors and services that are offering identity federation options for connecting your on-premise directory to business partners and applications in the cloud.

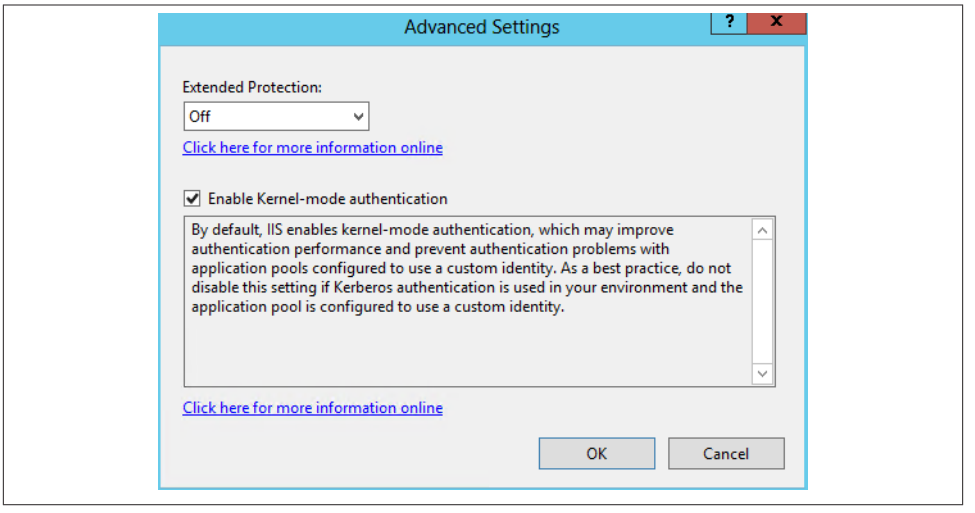


Figure 21-41. Extended Protection settings

With the name of this product including "Active Directory," it's a natural assumption that AD domain administrators will be able to take on ADFS as an extension of their duties. In reality, the skillset required to manage and troubleshoot ADFS is much more closely aligned with a web developer's role than a domain administrator's. This chapter is our attempt to bridge that gap and help get you, the domain administrator, on your way to delivering identity federation with ADFS in your organization.

APPENDIX A

Programming the Directory with the .NET Framework

Why bother learning about the .NET Framework? After all, many AD administrators have been scripting happily for years with VBScript, PowerShell, ADSI, WMI, and a pile of command-line tools. We have been getting along just fine. This .NET stuff is really for the enterprise developers writing line-of-business applications, is it not?

First, let's be clear that .NET may not be for everyone. There are many techniques available for programming the directory, and you may not need to move away from the tools and techniques you already use. However, there are some compelling reasons to consider it:

- Powerful features that were once only available to C++ developers are now being exposed in the .NET Framework. This makes it easier for the vast majority of us who do not program in C++ to get to these features.
- Microsoft has a powerful web development platform called ASP.NET that vastly improves how we can build applications for the Web. Many AD development tasks lend themselves to web-based deployment.
- Microsoft's 2008, hardly new! PowerShell command shell, based on the .NET Framework, completely changes the equation for how shell programmers and scripters can program and administer Windows.

Choosing a .NET Programming Language

.NET is a programming environment that can be accessed from many different languages. In fact, the runtime environment for .NET (or "managed") code is called the *Common Language Runtime*, or CLR. The runtime is fundamentally object-oriented, but is otherwise neutral to the language syntax used to program it. At the time of this

writing, there are dozens of different languages that can be used to write .NET code, from the time-honored COBOL to experimental languages like F#.

Microsoft itself ships tools for several of these languages, including C#, Visual Basic.NET (or VB.NET), C++, and JScript.NET. It is fair to say that most .NET developers use either C# or VB.NET, with Microsoft itself using C# for the vast majority of its own work. C# as a language looks a bit like C++, but a lot more like Java. However, for many of us, our background is in writing VBScript code, so for this book we will present all of the examples in VB.NET.

This decision will likely make a few people grumpy, but it is the best compromise. For those who are put off by this decision, please take consolation in the fact that the thing that really matters is learning the framework itself. The language details are not usually that important. In fact, the languages are often so similar that there are many free tools that can translate code from one language to another with no loss at all.

Unfortunately, there is no way we can explain the basics of programming .NET or any specific languages feature's in the space we have remaining. Those sorts of primers already exist on the Internet and in bookstores. Instead, we will take the approach of keeping things simple enough to not require a lot of background and additional depth.

Choosing a Development Tool

Since the last version of this book was published, the landscape for .NET development tools has expanded as much as the framework itself. Today, you have a lot of choices for how you can write .NET code. The primary decision to make is whether or not you want to use an integrated development environment (IDE) for your development tasks. IDEs simplify development by making it easier to organize your code files and build them into executables.

.NET IDE Options

Back in the days of Visual Studio.NET 2002, Microsoft was basically the only game in town for .NET IDEs. However, that story has changed significantly. Not only does Visual Studio 2012 come in a variety of packages, some of them completely free, but there are other non-Microsoft products available to choose from, such as SharpDevelop from IC# Code.

The point here is that there are many options, and quite a few of them are free. It is not necessary to buy an expensive tool to write .NET code. Although we do not endorse a particular tool, we will say that most .NET developers use Microsoft's Visual Studio, and the free Express versions are suitable for our needs, so that is a reasonable place to start.

.NET Development Without an IDE

Although an IDE can be nice, we do not actually need one to write code for the .NET Framework. In fact, Microsoft provides a free downloadable SDK with all versions of the framework that includes all the tools we need to build .NET code. Though it is rarely done, it is certainly possible to dive right in with Notepad and a command prompt! There are also a variety of tools available, such as Jeff Key's Snippet Compiler, that provide an experience similar to scripting, but with compiled code. In addition, all of the examples in this appendix can be accomplished in PowerShell since Windows PowerShell is built on top of the .Net Framework.

.NET Framework Versions

At the time of this writing, Microsoft has released versions 1.0, 1.1, 2.0, 3.0, 3.5, 4.0, and 4.5 of the .NET Framework. It can be confusing to understand which .NET features are available in which releases, which release came bundled with which operating system release, and how code written for one version will work on another version. Let's unravel all that.

First, a few general notes on .NET Framework versions:

- It is not a problem to install multiple versions of the framework on the same machine. They happily coexist when installed side by side. In fact, both .NET 3.0 and 3.5 require .NET 2.0 to be installed, because they are not actually standalone releases. NET 3.5 also requires Service Pack 1 for .NET 2.0 to be installed.
- By default, code compiled against a specific version of the framework will run against that same version of the framework if it is installed on the machine on which the code is run. However, if that version of the framework is not installed where the code is running but a newer version *is* installed, the code will run against the newer version.
- Generally speaking, code written against an earlier version of the framework will run with no problems against a later version of the framework. There are a few backward-compatibility problems, but they are rare overall.
- Make sure you install the latest service packs as they are released.

Which .NET Framework Comes with Which OS?

Depending on the operating system (OS) of the machine where you want to run .NET code, a version of the framework may already be included. Table A-1 summarizes which OS has what. An article at microsoft.com has an excellent diagram that shows the iterative versions of the framework and what version of the underlying CLR is used.

Table A-1. .NET Framework/Windows OS cross-reference

Operating system	Included framework versions
Windows 2000	None
Windows XP	None
Windows Server 2003	1.1
Windows Server 2003 R2	1.1, 2.0
Windows Vista	2.0, 3.0
Windows Server 2008	2.0, 3.0
Windows 7	2.0, 3.0, 3.5 SP1
Windows Server 2008 R2	2.0, 3.0, 3.5 SP1
Windows 8	4.5
Windows Server 2012	4.5

You can optionally install .NET 3.5 (along with .Net 2.0 and .Net 3.0) on Windows 8 and Windows Server 2012.

Unfortunately, once we include service packs and other factors, the matrix of possible options for .NET Framework installation across all of the different OS versions can get a little hairy. We will not attempt to document this completely and will instead defer to Microsoft's own deployment guides for the details:

- .NET 1.0: *http://msdn.microsoft.com/en-us/library/ms994336.aspx*
- .NET 1.1: *http://msdn.microsoft.com/en-us/library/ms994339.aspx*
- .NET 2.0: *http://msdn.microsoft.com/en-us/library/aa480237.aspx*
- .NET 3.0: *http://msdn.microsoft.com/en-us/library/aa480173.aspx*
- .NET 3.5: *http://msdn.microsoft.com/en-us/library/cc160717.aspx*
- .NET 4.0: *http://msdn.microsoft.com/en-us/library/ee390831(VS.100).aspx*
- .NET 4.5: *http://msdn.microsoft.com/en-us/library/ee390831.aspx*

If you're using .NET 3.5 with a forest that's at the Windows Server 2008 R2 functional level, make sure you've reviewed the hotfix described in Microsoft Knowledge Base article 2260240 to see if it applies to your application.

Directory Programming Features by .NET Framework Release

The .NET Framework includes a huge number of classes and other programming types. In order to keep things organized, these types are arranged in hierarchical groupings called *namespaces*. The core features of the framework are arranged under the `System` namespace.

There are now four main namespaces specifically used for directory services programming:

- `System.DirectoryServices` (SDS)
- `System.DirectoryServices.ActiveDirectory` (SDS.AD)
- `System.DirectoryServices.Protocols` (SDS.P)
- `System.DirectoryServices.AccountManagement` (SDS.AM)

For brevity, we will refer to the namespaces by the above abbreviations throughout the text.

This does not include all of the other types we might need to access to perform additional programming tasks and also ignores access to WMI (something that .NET can do, but not something we will cover at all in this book).

Assemblies Versus Namespaces

In order to keep the framework modular, major pieces of functionality are compiled into separate components called *assemblies*. An assembly is essentially just a DLL. An assembly can contain types in multiple namespaces, and namespaces can span different assemblies. Do not let this confuse you, though.

There are three important things to know here:

- In order to use a type in a specific assembly, our code must reference that assembly.
- The directory services programming features are bundled into separate assemblies from the rest of the .NET Framework so that code that does not need these features does not need to load the types contained in these assemblies.
- By default, none of the various projects types in Visual Studio reference *any* of the directory services programming assemblies, so we must remember to add these references. This will likely apply to whatever tool you end up using for development.

Summary of Namespaces, Assemblies, and Framework Versions

Table A-2 summarizes which namespaces are available in which .NET Framework releases.

Table A-2. Directory services namespaces by framework release

Namespace	Assembly	Version
`System.DirectoryServices`	*System.DirectoryServices.dll*	1.0+
`System.DirectoryServices.ActiveDirectory`	*System.DirectoryServices.dll*	2.0+
`System.DirectoryServices.Protocols`	*System.DirectoryServices.Protocols.dll*	2.0+
`System.DirectoryServices.AccountManagement`	*System.DirectoryServices.AccountManagement.dll*	3.5+

The bottom line is that .NET 2.0 contains the majority of the available features, but you need .NET 3.5 to get all of them. Looking at Table A-1, this also implies that you may need to get one of the newer versions of .NET installed where you want to run your code.

Our recommendation is that you use at least .NET 3.5, and we will assume .NET 3.5 unless otherwise noted. Use the latest service packs as well.

Directory Services Programming Landscape

First, let's take a high-level view of how all of the various pieces fit together.

As you might imagine, the .NET Framework does not attempt to implement the entire operating system from the ground up, but instead uses existing operating system services to some extent or another and repackages them in the more consistent .NET programming model. All of the .NET directory services features leverage existing operating system functionality, some of which we have already covered in this book.

Figure A-1 depicts how the various pieces fit together.

As with most software architectures, the model is built up in layers from low-level components to high-level ones. At the top, we see how the various .NET components fit together and depend on each other.

Right at the center of the diagram, we see ADSI and how it depends on various lower-level operating system components. You will find it reassuring that much of the investment you have already made in learning ADSI from this book and other sources will directly carry over to your .NET programming activities.

Now, let's dig into the high-level details of each namespace in turn.

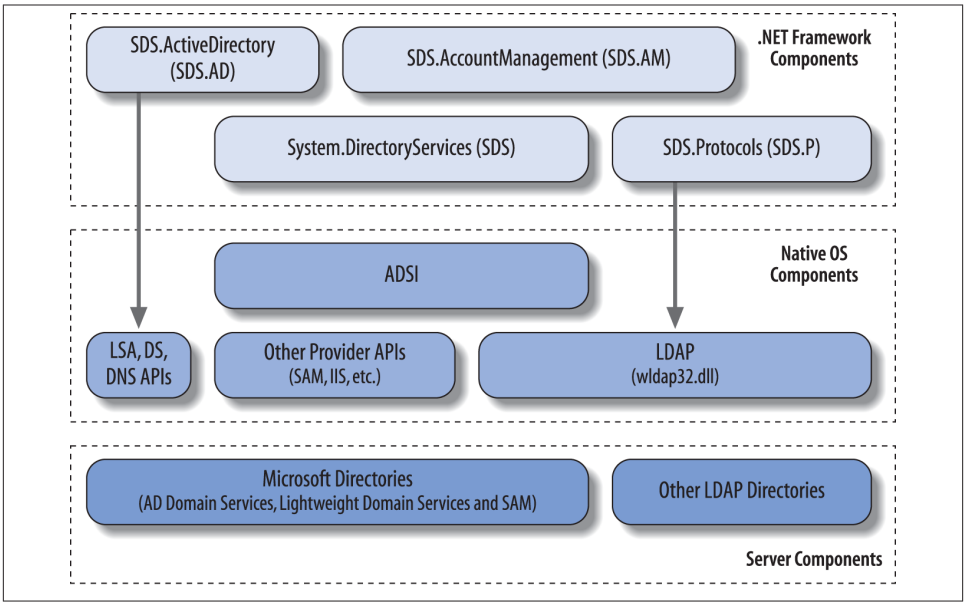


Figure A-1. Windows Directory Services API layers

System.DirectoryServices Overview

The `System.DirectoryServices` namespace (`SDS`) is the primary namespace we use for programming directory services in .NET. It is the oldest namespace, having been included with .NET since the very first 1.0 release.

`SDS` is basically just a simple interoperability layer over the existing ADSI programming model. Most of what you already know about ADSI will apply directly to `SDS`. `SDS` is in many ways like a stripped-down version of ADSI, but in other ways it is more powerful. In all cases, it is ADSI that does the bulk of the work, with the underlying components it depends on doing the actual heavy lifting.

By "stripped down," we mean that `SDS` supports the core interfaces in ADSI, such as `IADs` and `IADsContainer`, but does not directly support any of the so-called "persistent object interfaces," such as `IADsGroup` and `IADsUser`. This does not mean that we cannot use those interfaces, but that we may need to jump through some additional hoops to get to them.

Even though there are quite a few types in the namespace, almost everything in `SDS` revolves around two core classes:

- `DirectoryEntry`
- `DirectorySearcher`

`DirectoryEntry` is essentially a wrapper around the ADSI `IADs` interface and is most similar to programming with the familiar `IADsOpenDSObject.OpenDSObject` and `GetObject` methods. In fact, `DirectoryEntry` allows us to "drop down" into ADSI whenever we need to by exposing a `NativeObject` property that provides direct access to the underlying ADSI COM object.

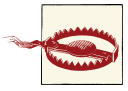
`DirectoryEntry` is used to connect to the directory, refer to specific objects, read and write their attributes, create new objects, delete objects, and move objects. Everything starts with a `DirectoryEntry` object.

`DirectorySearcher` is the more interesting part of `SDS`. As its name implies, it allows us to search the directory via LDAP queries. In the ADSI scripting world, we use ActiveX Data Objects (ADO) for querying the directory via the ADSI OLE DB provider, while we use `GetObject` to access specific objects and perform modifications. This approach made it easier for developers accustomed to querying SQL databases to learn LDAP, but the two models were never well integrated, and ADO gets a little clunky when special features need to be accessed. Additionally, many of the powerful advanced LDAP query features supported by Active Directory never got exposed to ADO, so developers using this technology were basically out of luck.

`DirectorySearcher` takes a different approach. Instead of using a similar pattern and having .NET developers use the classes in `System.Data` to execute LDAP searches, the designers of the framework gave us a component purpose-built for doing LDAP queries. The main advantages are:

- It offers tight integration with `DirectoryEntry` and the other supporting classes in `SDS`.
- It works in terms of LDAP-specific search concepts such as search base, query scope, and filters instead of using SQL terminology that may not apply.
- Special features such as paged searches and timeout settings are cleanly supported via strongly typed properties.
- All of the advanced Active Directory search features exposed by ADSI are supported.

You'll see how some of this looks when we show some examples in the upcoming sections.



The .NET Framework data access features in `System.Data` still support OLE DB data sources. This means that the OLE DB provider we used with ADO in VB and VBScript can still be used with .NET as well. This may seem like the natural thing to do for programmers coming from that background. Don't do it!

Besides having all of the same clumsiness issues that ADO always suffered from, it is not as fast and is completely unnecessary now that we have `DirectorySearcher`.

While we are issuing warnings, VB.NET also supports a `GetObject` method for backward-compatibility reasons. Don't use this for accessing Active Directory objects either. Use `DirectoryEntry` instead.

Under the hood, `DirectorySearcher` accomplishes its magic by interoperating with a low-level ADSI interface called `IDirectorySearch`. `IDirectorySearch` has been in ADSI for a long time, but VB and VBScript developers could not call it directly as it was designed only for C++ developers. The ADSI OLE DB provider we use in VBScript is actually another wrapper around `IDirectorySearch` that adapts `IDirectorySearch` to the ADO programming model. With `DirectorySearcher`, we get a feature designed for LDAP programming and with all the special features exposed.

Other nice things in System.DirectoryServices

The .NET Framework has first-class support for Windows security objects such as security descriptors, DACLs, ACEs, and SIDs. Consequently, `SDS` follows suit and provides support for the full complement of Active Directory security-descriptor-manipulation functions. The classes that support this are much faster and more powerful than the ADSI-based interfaces we used to program them before, such as `IADsSecurityDescriptor`.

We usually do not spend much time writing code that manipulates security descriptors, but full support for these features is welcome in the cases when we do.

System.DirectoryServices summary

- `SDS` is the primary namespace we use for .NET directory services programming.
- `DirectoryEntry` and `DirectorySearcher` are the two main classes.
- `SDS` is based on ADSI, so much of our ADSI knowledge applies directly.
- `SDS` is a little stripped-down compared to full ADSI, but it works well in general.
- `DirectorySearcher` is a better query interface than what we had before.

System.DirectoryServices.ActiveDirectory Overview

As its name implies, `SDS.AD` provides support for performing operations on Active Directory specifically. When we say Active Directory, we include both AD Domain Services and AD Lightweight Directory Services (AD LDS).

`SDS.AD` provides access to functions that previously were difficult or impossible for developers using languages other than C++ to access. The functions can be grouped into the following categories:

- Active Directory infrastructure (forests, domains, global catalogs, domain controllers, sites, etc.)
- Replication
- Trusts
- Directory schema

`SDS.AD` accomplishes this goal by using a clever design to marry the ADSI programming model to a set of non-LDAP RPC interfaces with function names like `DsGetDCName`. As we will see, `SDS.AD` is integrated tightly with `SDS`, making it easy to move between the two namespaces as needed.

Why use System.DirectoryServices.ActiveDirectory?

You may be looking at the list from the previous section and thinking that the number of times you've needed or wanted to be able to write code to create a trust could be counted on one hand (or less), and you might be right. In fact, much of the functionality exposed by `SDS.AD` is stuff that most of us will never need to automate. The tasks involving `SDS.AD` classes are infrequent, and other tools are often more appropriate to accomplish those tasks.

The sweet spot for `SDS.AD` is in the first point, which is the access to Active Directory infrastructure components. The ability to do things like enumerate domains and domain controllers is helpful and very easy to use here. Additionally, `SDS.AD` provides access to the full power of the DC locator component of Windows (discussed in Chapter 8) and allows us to access all of its advanced features, such as finding domain controllers in specific sites and forcing rediscovery. This was never available to us at all via ADSI or `SDS` before.

With this in mind, we will proceed to ignore all of the other, more obscure features of `SDS.AD` and instead concentrate on the infrastructure features in our upcoming examples.

System.DirectoryServices.ActiveDirectory summary

- `SDS.AD` provides many powerful features that were not previously available to most programmers.
- It is tightly integrated with `SDS`.
- Some of the functions supported are a bit obscure, but the infrastructure features are always useful.

System.DirectoryServices.Protocols Overview

Unlike `SDS.AD`, which layers new capabilities on top of the existing ADSI-based `SDS`, `SDS.P` is a completely different animal. Referring back to Figure A-1, we see that `SDS.P` does not layer on top of ADSI at all, but instead sits directly on top of the core Windows LDAP library, *wldap32.dll*. We also see that `SDS.P` does not overlap with `SDS` at all.

In a nutshell, `SDS.P` gives us a completely different model for programming LDAP. Instead of using the "object-centric" metaphor of ADSI, where you create programmatic objects that point to specific objects in the directory in order to manipulate them, `SDS.P` applies the "connection-centric" metaphor inherent in the standards-based LDAP API design. For example, in `SDS.P` we create a `DirectoryConnection` object to connect to the underlying directory, and then we perform operations on that connection by sending it messages such as search requests and receiving messages such as search responses in reply.

`SDS.P` implements just about every feature of the underlying LDAP API; it brings the full power of LDAP to the programmer, including many features not even supported by ADSI and thus not possible in `SDS`. It also provides the opportunity to get the best possible performance from LDAP in scenarios where performance trumps productivity (i.e., not usually scripting).

The downside of this is that `SDS.P` is more complicated and demands that you learn all of the intimate details of LDAP and AD programming. Basically, it is provided to support the needs of systems programmers rather than administrators or even normal enterprise developers.

Why use System.DirectoryServices.Protocols?

For most of us, the answer is that we will probably never need to. This namespace is not for us, so we can safely ignore it.

However, there are definitely specific scenarios where `SDS.P` really is needed or desirable, and we can at least list a few of them:

- You need access to obscure AD LDAP features not supported by ADSI, such as the phantom root or stats controls.
- You are programming against a non-Microsoft LDAP directory and ADSI does not mesh well with it.
- You are writing a highly scalable multithreaded server application and need full support for asynchronous queries and maximum performance (not available with ADSI).
- You use SSL with your LDAP programming and need to override the SSL certificate verification policy so that certain error conditions that would cause an ADSI connection to fail can be ignored.
- You need to query a directory using the Directory Services Markup Language (DSML) protocol instead of the LDAP protocol.

At the same time, the difference in complexity between writing AD code in VBScript and LDAP API code in C++ is much larger than the difference between programming `SDS` and `SDS.P` in VB.NET. If you are inclined to get your hands dirty and are pretty comfortable with LDAP, feel free to check it out. We will provide one simple sample and leave it at that.

System.DirectoryServices.Protocols summary

- `SDS.P` provides a powerful, low-level library for programming LDAP directly (no ADSI).
- It is not intended for use by typical administrators and enterprise developers; it is intended for systems programmers.
- In specific situations, it may be the only way to accomplish a particular goal, but most of these scenarios are uncommon.
- It is probably safe for most of us to ignore it.

System.DirectoryServices.AccountManagement Overview

Since its introduction in NET 1.0, `SDS` has often been criticized for being too difficult to use for performing common user- and groups-management tasks. It was seen as a step down from the approach provided by ADSI before it. Microsoft did not really do anything to address this shortcoming in .NET 2.0, but the 3.5 release changed this significantly.

In .NET 3.5, Microsoft introduced a new namespace and assembly called `SDS.AM`. Also called "the principal API," `SDS.AM` significantly simplifies the tasks associated with managing directory security principals such as users, computers, and groups.

These are the design goals of SDS.AM:

- Provide a unified design for managing security principals that works the same across all three of Microsoft's primary directory platforms: AD Directory Services, AD Lightweight Directory Services, and the local machine Security Accounts Manager (SAM) database.
- Reduce the need to understand LDAP programming or the underlying details of the store being accessed.
- Make the design flexible so that it can be extended to provide custom functionality and "drop down" into SDS whenever more power is needed.

Referring back to Figure A-1, we see that SDS.AM layers on top of SDS, with a bit of overlap with SDS.P as well. For the most part, SDS.AM is based on SDS and thus ADSI. In a way, we could say that SDS.AM reintroduces features ADSI always had with the IADsUser, IADsComputer, and IADsGroup interfaces, but in actuality it does ADSI one better. Because SDS.AM is much newer, it allowed the designers to purposefully include support for AD LDS and to take advantage of many .NET programming features that simplify development tasks. We will see this when we dive into the examples.

Referring to Figure A-2, we see that there are just a few primary classes within SDS.AM. On the left, we see a class called Principal with several other classes underneath it. The lines here are significant because they indicate an inheritance relationship. UserPrincipal derives from AuthenticablePrincipal, which derives from Principal, etc. Principal is essentially the center of the universe in SDS.AM. We can also derive our own classes from any of these classes as a way to extend SDS.AM to our own directory customizations.

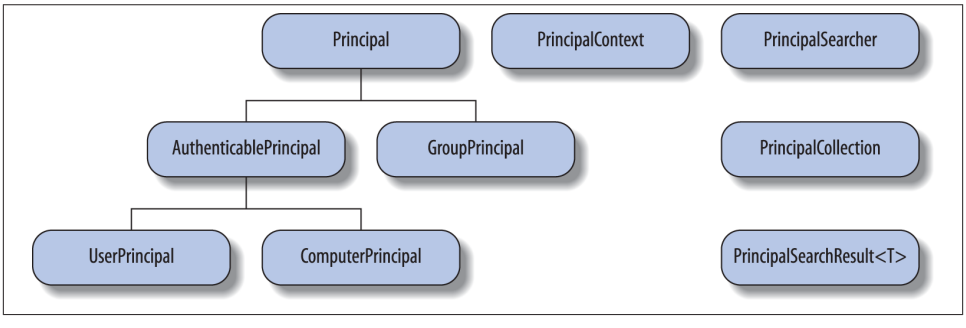


Figure A-2. Primary classes in System.DirectoryServices.AccountManagement

PrincipalContext is a helper class used to create different types of Principal objects. Its primary job is to specify which type of directory we are accessing.

`PrincipalSearcher` and the classes underneath it are used for finding `Principal` objects in the directory and work in conjunction with `PrincipalContext` and `Principal`.

Why use System.DirectoryServices.AccountManagement?

Unlike `SDS.P`, `SDS.AM` really is intended to be used by most of us. It significantly simplifies many common tasks and is much easier to use than any of the other directory services programming namespaces. It provides such a high degree of abstraction that it makes it possible for a programmer to know almost nothing about LDAP, ADSI, or the underlying directory model to accomplish common tasks. It is by far the most productive API we have been given to date. It also contains a comparatively small set of types in the namespace, so it requires little effort to get started.

System.DirectoryServices.AccountManagement summary

- `SDS.AM` is a namespace designed specifically for managing security principals in Microsoft directories.
- It requires less knowledge about LDAP and the underlying directories than any of the other namespaces.
- It provides a highly productive developer experience.

.NET Directory Services Programming by Example

Now, let's switch focus and dive into some code. Unfortunately, there is no possible way we can do much more here than scratch the surface of all of the types of things we can do with .NET directory services programming. This topic could easily fill an entire book (and has a few times already). If you need a deeper reference, *The .NET Developer's Guide to Directory Services Programming* by Joe Kaplan and Ryan Dunn (Addison-Wesley) is an excellent option.

Here are the conventions for the code samples:

- For brevity, we will skip error handling and object cleanup code that we really should be using in most "real" work.
- Assume that we are using a simple console application type for our samples, but do not let that imply that you can only build console applications. All of these samples may be used in any .NET application type.
- Assume we have an assembly reference to *System.DirectoryServices.dll* and have imported namespaces for `SDS` and `SDS.AD` via the following block of code:

```
Imports System.DirectoryServices
Imports System.DirectoryServices.ActiveDirectory
```

Like VBScript and VB6, VB.NET supports a programming technique called "late binding." Late binding allows us to refer to properties or methods on an object that may not be explicitly available on the current class. This is mostly helpful when working with COM interfaces in .NET.

However, C# does not have such a feature and requires complex "reflection" code to accomplish the same goal. For that reason, we will avoid using any late binding in our samples. `SDS` provides other ways to accomplish the same tasks anyway, so late binding is not really necessary. You can add the `Option Strict` and `Option Explicit` directives in your code to disable late binding if you wish.

Additionally, VB.NET is not case-sensitive but C# is, so we'll use proper case in all of our examples.

Connecting to the Directory

There are two basic options available for connecting to the directory:

- Use `DirectorySearcher`.
- Use one of the infrastructure classes in `SDS.AD`.

Let's look at some examples of each in turn. As we discussed earlier in the book, LDAP directories support a special object called *RootDSE* that allows the programmer to query the directory for information such as what partitions it contains, so we will start there:

```
'connect to rootDSE using "serverless binding"
Dim rootDSE As New DirectoryEntry("LDAP://RootDSE")

'this uses the full constructor to do the same thing
Dim rootDSE2 As New DirectoryEntry( _
    "LDAP://RootDSE", _
    Nothing, _
    Nothing, _
    AuthenticationTypes.Secure _
    )

'same as first but specifying a specific domain
Dim rootDSE3 As New DirectoryEntry( _
    "LDAP://mycorp.com/RootDSE")

'now we are using explicit credentials and
'Secure (negotiate) authentication
```

```
Dim rootDSE4 As New DirectoryEntry( _
    "LDAP://rootDSE", _
    "someuser@mycorp.com", _
    "Password1", _
    AuthenticationTypes.Secure _
    )

'bind to LDS rootDSE as an AD LDS user using
'simple bind and SSL to protect the credentials
'on the network
Dim ldsRootDSE As New DirectoryEntry( _
    "LDAP://adldsserver.com/RootDSE", _
    "otheruser@adlds", _
    "Password1", _
    AuthenticationTypes.SecureSocketsLayer _
    )
```

The first thing to notice is that this looks quite a bit like `GetObject` or `OpenDSObject` in ADSI. This is no accident. The same technology is being used and the same rules apply. In .NET, we use a feature called "method overloading" to specify multiple constructors for the same object. This makes it easy to either use the defaults or specify explicit credentials and other options whenever we need to. We also show an example of connecting to AD LDS at the end. Again, the same rules apply, except that we cannot use "serverless binding" with AD LDS and must specify a server name.

In the AD LDS example, we connect using SSL and use LDAP simple bind instead of Windows secure bind.

There is no flag to specify a simple bind. Instead, we supply non-null credentials and specify an `AuthenticationType` other than `Secure`. It is a little confusing at first, but it actually works the same way in ADSI. If we still want to do a secure bind with a Windows user and also use SSL, we simply combine the flags, like this:

```
AuthenticationTypes.Secure Or AuthenticationTypes.SecureSocketsLayer
```

There are many other flags available, but we simply do not have room to explain them all. They are the same flags available in ADSI in the `ADS_AUTHENTICATION_ENUM`, just with slightly different names that use .NET standards. Reference *microsoft.com* for a listing of possible `AuthenticationTypes`.

Now, let's read some attributes and connect to the domain root object:

```
'get the defaultNamingContext for this domain
Dim ncName As string = DirectCast( _
    rootDSE.Properties("defaultNamingContext").Value, _
    String)
Dim ncRoot As New DirectoryEntry("LDAP://" + ncName)

'now, let's use SDS.AD to do the same thing
'in one line of code!
Dim ncRoot2 As DirectoryEntry = _
    Domain.GetCurrentDomain().GetDirectoryEntry()
```

In this example, we show how to read LDAP attributes from the `DirectoryEntry` we created previously. Instead of using `Get` and `GetEx` like we do in ADSI, .NET implements a `PropertyCollection` that contains `PropertyValueCollection` objects. The `Value` property on `PropertyValueCollection` is a helper that returns:

- A null reference (`Nothing`) if the attribute is not set or not available
- A single value as `Object` if the attribute has a single value
- An array of `Object` values if the attribute is multivalued

This is handy, so we will use it frequently. Since attributes in AD can be of many different data types and .NET converts them to specific data types the same way that ADSI does, they are returned as `Object`. This means we must convert them back into the type they really are.

After that, we use the `defaultNamingContext` attribute value from `RootDSE` to build another `DirectoryEntry`, this time pointing to the domain partition root object.

Next, we use an alternate approach to accomplish the same goal in a single line of code using the `Domain` class. Obviously, this is less work and quite handy. Here are some other examples with the `Domain` class:

```
'create a Domain-type DirectoryContext pointing to
'a specific domain with default credentials
Dim context1 As New DirectoryContext( _
    DirectoryContextType.Domain, _
    "othercorpdomain.com" _
    )
'build a Domain object with the context
Dim domain1 As Domain = Domain.GetDomain(context1)
'create a Domain-type DirectoryContext pointing to
'a specific domain with alternate plaintext credentials
Dim context2 As New DirectoryContext( _
    DirectoryContextType.Domain, _
    "othercorpdomain.com", _
    "someuser@mycorp.com", _
    "Password1" _
    )
Dim domain2 As Domain = Domain.GetDomain(context2)
```

With SDS.AD, we use an object called DirectoryContext as a way to build all sorts of different types of objects, such as domains, forests, specific servers, and even AD LDS configuration sets or AD application partitions. We see this pattern repeated in SDS.AM with the PrincipalContext class.

We do not set AuthenticationTypes with DirectoryContext in the same way we can when building a DirectoryEntry. The underlying code sets all the right defaults for us, including Signing and Sealing, although there is no option to use either SSL or simple bind. SDS.AD only supports Windows authentication for both AD DS and LDS.

As we said before, SDS.AD provides a vast array of AD management features, and we do not have room here to even scratch the surface. Some of our favorite features include the ability to expand the complete domain tree and locate Global Catalog servers, from which you can directly construct a DirectorySearcher for searching the GC. We'll end with one final example showing how to enumerate domain controllers in a specific domain, since nearly everyone has to do this from time to time:

```
Dim currentDomain As Domain = Domain.GetCurrentDomain()
For Each dc As DomainController _
    In currentDomain.FindAllDomainControllers()
    'we can do whatever we want now that we have all the DCs...
    Console.WriteLine(dc.Name)
Next
```

The example is trivially easy, and that is really the whole point. SDS.AD takes tasks that can range from annoying to nearly impossible and makes them easy to accomplish.

The IDisposable Interface and Directory Services Programming

The .NET Framework provides an interface called IDisposable that is applied to many objects throughout the framework. IDisposable has a single method called Dispose, with the intended use being that developers are encouraged to call the Dispose method on objects when they are done with them rather than allowing the .NET runtime to cleanup these objects through the normal garbage collection mechanism. The reason this is sometimes necessary is that .NET objects that interoperate with native resources via COM and direct API calls often hold references to memory and other operating system resources that are not managed directly by the runtime. Even though well-designed objects that also implement a Finalize method will eventually cause these resources to be released when the runtime's garbage collector gets around to calling it, that may take longer than would be desirable otherwise, especially in cases where small .NET objects reference expensive native resources.

Developers may either call the `Dispose` method directly or use the `Using` statement (variants of which are available in both VB.NET and C#) to do this automatically. The `Using` statement is the preferred method and looks like this:

```
Dim entry As New DirectoryEntry("LDAP://RootDSE")
Using (entry)>
    'do something with the DirectoryEntry here...
End Using
```

The reason we mention this is that many objects in the .NET directory services programming namespaces implement the `IDisposable` interface, including `DirectoryEntry`, `DirectorySearcher`, and `SearchResultCollection`, and significant chunks of the types in `SDS.AD`, `SDS.AM`, and `SDS.P`. This should come as no surprise, as we have seen how much all these namespaces depend on underlying native APIs.

Even though our code samples do not demonstrate this, it is always a good idea to wrap your `IDisposable` objects in a `Using` block. In short-lived programs, the difference is likely to be minimal, but it can make a huge difference in resource usage for applications that instantiate many of these objects or with long-lived processes such as those that are typical in .NET web applications.

We said that these objects will eventually get cleaned up by the garbage collector one way or the other, and for the most part that is true. However, in the case of the `SearchResultCollection`, it is not. It currently has a limitation that does not allow the `Finalize` method that is called by the garbage collector to release the underlying native resource. As such, you should always wrap `SearchResultCollection` in a `Using` block.

Searching the Directory

One of the primary reasons we have a directory in the first place is to look up information in it, so it stands to reason that searching the directory is one of the things we'll do most often. Let's look at some basic examples of searching with `DirectorySearcher`:

```
Dim root As DirectoryEntry = _
    Domain.GetCurrentDomain().GetDirectoryEntry()
Dim searcher As New DirectorySearcher(root)
searcher.Filter = "(&(objectCategory=person)(objectClass=user))"
searcher.SearchScope = SearchScope.SubTree

'we only want this attribute returned, so specify it
searcher.PropertiesToLoad.Add("sAMAccountName")

'enable paging by setting a nonzero page size
searcher.PageSize = 1000
Dim src As SearchResultCollection
Using (src)
    src = searcher.FindAll()
    For Each result As SearchResult In src
        Console.WriteLine(result.Properties("sAMAccountName")(0))
```

```
    Next
End Using
```

A `DirectorySearcher` object always needs a `DirectoryEntry` to use as a way of establishing the connection to the directory and determining which object will be the base of the search. In this example, we use our previous shortcut with `SDS.AD` to build a `DirectoryEntry` pointing to the domain partition root object for this purpose. Note that you can build a `DirectorySearcher` without specifying this, and `SDS` will try to do this for you, but it is nearly always better to be explicit about your intent in your code.

We use a standard filter for finding user objects and set the scope to `SubTree`, even though that is not technically necessary since it is the default. We also add the `sAMAccountName` attribute to the list of `PropertiesToLoad` to specify that we want only that attribute returned instead of the default of "all," as per normal LDAP rules.

From a performance perspective, it is important to supply values to `PropertiesToLoad` as a normal procedure. If we do not, the directory will return Active Directory default attribute values for each matched object. This results in a significant waste of network bandwidth, especially for searches that return many results. It is also worth noting that if we wish to return operational or constructed attributes such as `canonicalName` or `allowedAttributesEffective`, we must specify them in `PropertiesToLoad` as they are never returned in a default search.

We also set the `PageSize` to `1000` to enable paged searches. As with other ADSI-based searches, we do not explicitly loop through the pages of results returned by the server under the hood. ADSI does this for us. Generally speaking, it is always a good idea to enable paged searches as we usually want to get all of the results, even if there are more than 1,000. If there are less than 1,000, the paging request does not hurt anything.

To enumerate the results, we use a `For Each` loop against the `SearchResultCollection` object returned by the call to `FindAll`. Note that this approach provides better performance than using a `For` loop based on the `Count` property of the `SearchResultCollection`. This has to do with the fact that in order to get the `Count`, the entire search must be run under the hood anyway. Using `For Each` instead fetches the results as they are returned.

Notice that like `DirectoryEntry`, the `SearchResult` class also has a `Properties` property that provides access to the attribute values returned by the search. They behave similarly, except that `SearchResult` objects are strictly read-only and the `ResultPropertyValueCollection` does not have a handy `Value` property like the `PropertyValueCollection`, so we must access the first value returned by indexing into the array. Make sure that if you access an attribute that might be null, you first check to see if the

SearchResult contains that attribute using the Contains method. Otherwise, indexing into the array will result in an error:

```
If result.Properties.Contains("displayName") Then
    Console.WriteLine(result.Properties("displayName")(0))
End If
```

The SearchResult class contains a method called GetDirectoryEntry that will return the DirectoryEntry object for any given result using the same security context and connection information supplied to the original DirectoryEntry used to build the SearchRoot for the DirectorySearcher. While it may seem tempting to use this method to access the property cache to read attributes from the search, this is a bad idea from a performance perspective. The reason is that creating a DirectoryEntry object results in at least one and usually two additional searches of the directory to fill the property cache for each object. Since the SearchResult object already contains the results we asked for in our search, it makes no sense to access this data again in a less efficient way. Always use the SearchResult when you need to read attribute values from a search.

The appropriate case to call the GetDirectoryEntry method is when you need to perform modifications on the returned object. SearchResult objects are read-only, so we must have a DirectoryEntry if we need to perform a modification.

DirectorySearcher is also the place where many of the special features in AD LDAP are lurking. For example, DirectorySearcher provides access to things like attribute scope queries, directory synchronization, virtual list views, deleted object searches, and *Extended DN Queries*. Once again, we are pressed for space in this book, but check out some of the other resources available to find out how to use these advanced features if you find yourself in need of them.

Basics of Modifying the Directory

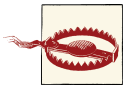
Now that we have some of the basics covered, let's turn our attention to modifying the directory. After all, searching is only useful if someone puts some objects in the directory first.

We have a few options when it comes to modifying the directory. SDS allows us to create, modify, or delete just about any type of object in the directory that we can think of, as long as we know how. It is our workhorse for modifications, so we will start with it.

On the other hand, much of an administrator's life tends to revolve around modifying security principals in the directory, such as users, computers, and groups. As we've already learned, the namespace in introduced .NET 3.5, SDS.AM, gives us a simple,

powerful way to manipulate those types of objects specifically. In the next section, we will show some samples for performing those tasks, using both namespaces for contrast.

Finally, `SDS.AD` again provides us with a rich set of features for modifying AD infrastructure objects such as sites, trusts, and the schema (which is almost always a bad idea to modify in code, but this is sometimes necessary). This essentially allows you to build your own versions of the AD Domains and Trusts, Sites and Services, and Schema Management MMC snap-ins, if you so desire. While we won't look at these types of scenarios in depth, we do want you to know where to look if the need arises.



You cannot perform modification operations on a read-only domain controller. If you attempt to do so, the DC will issue a write referral to a writable DC. This may or may not work.

The other problem with writes to RODCs is that if the write referral did work, the change would go to a different DC than the DC you were trying to access, so the original RODC would not have the new data until replication completes. This can cause chaos in an application that does not anticipate this.

By default, ADSI asks for writable DCs, so this may or may not actually be a problem. The best solution is to be explicit in your intentions and ask for a writable DC. Fortunately, .NET provides a clean way to do this. The `LocatorOptions` enumeration now provides a `WriteableRequired` value that can be specified, so make sure you use the overloaded versions of methods for locating DCs (such as `Domain.FindDomainController`) that allow you to specify `LocatorOptions.WriteableRequired` when needed.

In order for ADSI on machines running Windows XP or Windows Server 2003 to contact an RODC, you will need to install the compatibility package available from *http://support.microsoft.com/kb/944043*.

Basic add example

Let's get started with a simple example, adding a new OU under the root of the domain:

```
Dim root As DirectoryEntry = _
    Domain.GetCurrentDomain().GetDirectoryEntry()
Dim newOU As DirectoryEntry = _
    root.Children.Add("OU=New OU", "organizationalUnit")
newOU.Properties("description").Value = "A new OU"
newOU.CommitChanges()
```

Adding a new object to the directory using `DirectoryEntry` revolves around using the `Add` method on the `Children` property of the `DirectoryEntry` object that will be the parent of the newly added object. The first parameter of the `Add` method takes the relative distinguished name (RDN) of the new object. For a default Active Directory forest, this will always be `CN=`*xxx*, `UID=`*xxx*, `OU=`*xxx*, or `DC=xxx`, as those are the four default RDN

attribute IDs. With AD LDS, there is more flexibility. The second parameter indicates the `objectClass` of the new object.

The second thing to notice is that the `Add` method returns a new `DirectoryEntry` object. This object is not yet persisted in the directory, but instead exists in memory until we actually call the `CommitChanges` method. This allows us to set additional attributes on the object before it is first saved; it is important because some objects define mandatory attributes that must be set on the object at creation time, or optional attributes that can only be set at creation time (schema objects are an example here). Here, we show adding a `description` attribute to demonstrate how this works, although `description` is not mandatory and could be added after the fact.

Basic remove examples

Removing objects is simple. As a quick example, let's remove the OU we just created. We will reuse the same variables we already initialized:

```
root.Children.Remove(newOU)
```

Once again, we use the `Children` property on `DirectoryEntry`, this time calling its `Remove` method. The `Remove` method takes another `DirectoryEntry` object as its parameter, which indicates the object to be removed. As long as the object passed in is a child of the parent and we have the necessary permissions, the deletion happens immediately.

If we instead want to delete an object and all of its descendants, we must use a slightly different approach. For simplicity, let's pretend "New OU" from our creation example now has a bunch of child objects, including other containers with children:

```
newOU.DeleteTree()
```

Here, we call the `DeleteTree` method on the object that we wish to delete along with all of its descendants. This is a little different than the previous example, where we removed a single object via the `Remove` method on the parent object's `Children` property.

The overall point here is that we must know which type of deletion we want to perform and use the appropriate technique. In both cases, the deletion is immediate. Hopefully it goes without saying that we must be extremely careful when deleting objects in general!

Moving and renaming objects

In ADSI, we have a multipurpose method called `MoveHere` that we use for doing object moves, object renames, or both operations at the same time. `SDS` attempts to simplify this model by providing separate `MoveTo` and `Rename` methods instead.

`MoveTo` provides two overloads. One allows us to simply move the object, while the second overload allows us to move and rename it at the same time. Let's take a look. In

this example, we have two `DirectoryEntry` objects pointing to OUs in our domain, and we wish to move one OU under the other:

```
Dim firstOU As New DirectoryEntry( _
    "LDAP://OU=First OU,DC=mycorp,DC=com")
Dim newParent as New DirectoryEntry( _
    "LDAP://OU=Second OU,DC=mycorp,DC=com")

'choose one or the other!!!
'this one moves without renaming
firstOU.MoveTo(newParent)

'this one moves and renames
firstOU.MoveTo(newParent, "OU=Child OU")
```

In the last line, where we rename the object while moving it, we supply the new name in RDN format as we did in the creation sample. We also use this same approach when using the `Rename` method to change an object's name without moving it. Let's rename our second OU from the previous sample:

```
Dim toBeRenamed As New DirectoryEntry( _
    "LDAP://OU=Second OU,DC=mycorp,DC=com")
toBeRenamed.Rename("OU=A different name")
```

Modifying existing objects

The final aspect of modification basics is changing attribute values on existing objects. For the most part, this works exactly the same way we saw in our creation example, where we added an attribute value between the time we initially created the object and when we saved it to the directory. We set the desired attribute to the value we need and call `CommitChanges` when we are done:

```
Dim entry as New DirectoryEntry( _
    "LDAP://CN=test,OU=People,DC=mycorp,DC=com")

entry.Properties("displayName").Value = "new name"
entry.Properties("description").Value = "new description"
entry.CommitChanges()
```

Using the `Value` property replaces the existing value with the new value or sets the attribute if it was not set previously.

If the attribute is multivalued and we wish to modify incrementally instead of replacing the whole thing, we should instead use the `Add` and `Remove` methods to add and remove specific values. We can also replace an entire multivalued attribute using the `Value` property again. To remove all attribute values, we call the `Clear` method.

The other main thing to keep in mind is that we must supply data in the appropriate type for the attribute value being set. Most directory attributes are strings, but some

take numeric, date, or binary data, so check the schema reference documentation to know for sure.

Managing Users

Now that we have the basics of directory modifications mastered, let's look at a few user management examples. As we have learned by now, effective user management comes down to knowing many of the picky little details about how data is stored in the directory and how to change it to get the results we need. Let's start with a simple user creation sample:

```
Dim parent As DirectoryEntry = New DirectoryEntry( _
    "LDAP://OU=people,DC=mycorp,DC=com")
Dim user As DirectoryEntry = _
    parent.Children.Add("CN=test.user", "user")
user.Properties("sAMAccountName").Value = "test.user"
user.Properties("userPrincipalName").Value = "test.user@mycorp.com"
user.CommitChanges()
```

This looks fairly similar to our previous example of creating an OU, except that we set some different attributes for the user object. The problem with this is that this user will not have a password and will be disabled by default, so it is not very useful yet. Additionally, if we have a password policy in place requiring passwords, we cannot enable the user until after we have set a password. Let's revise our sample:

```
Dim parent As DirectoryEntry = New DirectoryEntry( _
    "LDAP://OU=people,DC=mycorp,DC=com")
Dim user As DirectoryEntry = _
    parent.Children.Add("CN=test.user", "user")
user.Properties("sAMAccountName").Value = "test.user"
user.Properties("userPrincipalName").Value = "test.user@mydomain.com"
user.CommitChanges()

'this is how we call an IADsUser method
user.Invoke("SetPassword", New Object() {"Password1"})
'this is how we call an IADsUser property method
user.InvokeSet("AccountDisabled", New Object() {False})
'or we could do this, which is faster:
user.Properties("userAccountControl").Value = 512 'normal

'force password change at next logon
user.Properties("pwdLastSet").Value = 0
user.CommitChanges()
```

Suddenly, our sample is getting a little complex. First, notice that we use a method on `DirectoryEntry` called `Invoke` to call methods on underlying ADSI interfaces. In this case, we call `IADsUser.SetPassword`. `Invoke` takes an array of objects as its other argument (although in this case it really just wanted a single string), because the method

we could be calling under the hood might take any number of different arguments of any type and we need a generic mechanism to make this work.

Next, we see a call to `InvokeSet`. While `Invoke` is used for calling ADSI methods, `InvokeGet` and `InvokeSet` are used specifically for calling the set and get versions of ADSI properties—in this case, `IADsUser.AccountDisabled`. We also show how you could set `userAccountControl` directly instead of using the ADSI property, although we cheat a little by setting a direct numeric value instead of changing the single bit we need to flip in order to remove the disabled flag. In general, setting this attribute to an arbitrary value is bad practice. Instead, we should manipulate the individual bit flags to avoid overwriting other settings and make our intentions in our code more clear.

Finally, we set the `pwdLastSet` attribute to `0` to force the user to change her password at the next login and call the `CommitChanges` method to update the object in the directory.

We also see that we have to do this whole thing in three steps. We cannot call `SetPassword` on an object that has not been created yet, and we cannot enable an object that has no password, so all three steps are needed.

Now, let's pretend we need to add a user to AD LDS. In this case, we use a totally different method to enable the user. Instead of using `userAccountControl`, AD LDS uses the `msds-userAccountDisabled` Boolean attribute (set to `False`, as you might guess). While this is certainly easier to deal with than flipping individual bits on `userAccountControl`, the problem is that we need different code for different stores and have to keep track of all these details.

Once we throw in all of the rest of the user management functions, such as setting the home directory, account expiration, and so on, it gets messy quickly.

Managing users with System.DirectoryServices.AccountManagement

Now, let's take a look at how `SDS.AM` attempts to simplify this by creating the same user again:

```
Dim principalContext As new PrincipalContext( _
    ContextType.Domain, _
    "MyCorp", _
    "OU=People,DC=mycorp,DC=com")
Dim newUser As New UserPrincipal( _
    principalContext, "test.user", "Password1", True)
```

Obviously, this is much simpler. The fourth parameter tells `SDS.AM` that we would like the user enabled during the creation. In fact, the only real LDAP thing we need to know here is the distinguished name of the container we want to put the user in.

Where it gets even nicer is the fact that we can create a user in either the local SAM database or AD LDS simply by varying how we build the initial `PrincipalContext` object. The second line of code stays the same:

```
Dim samContext As New PrincipalContext(ContextType.Machine)
Dim ldsContext As New PrincipalContext( _
    ContextType.ApplicationDirectory, _
    "localhost:50000", "OU=Users,O=Demo")
```

Unlike with `SDS`, where we use the `Invoke` and `InvokeSet` methods to call into the underlying ADSI interface members, `SDS.AM` provides strongly typed .NET properties and methods that allow us to manipulate users, groups, and computers in a friendlier way. For example, we now have methods such as `SetPassword`, `ExpirePasswordNow`, `IsAccountLockedOut`, and `UnlockAccount`, and properties such as `LastPasswordSet` and `UserCannotChangePassword`, to make all of these operations that were once difficult (or at least annoying) both simple and easy to understand.

The productivity gains with `SDS.AM` do not stop there. `SDS.AM` also provides:

- The full set of user and computer provisioning functions, including read, update, move, and delete functions
- A comprehensive set of group management functions, including provisioning of groups and expansion of group membership for both groups and users
- A set of functions for finding principals in the directory that require little, if any, LDAP knowledge
- An extensibility model that allows us to create our own `Principal` classes that support our own schema modifications or other relevant AD schema that are not directly supported in the "out-of-the-box" `Principal` classes
- A high-performance bind authentication feature for performing LDAP authentication in a scalable, dependable way

Again, we can only scratch the surface here. To dig a little deeper, check out the MSDN magazine article "Managing Directory Security Principals in the .NET Framework 3.5" by Ethan Wilansky and Joe Kaplan, from the January 2008 issue.

Overriding SSL Server Certificate Verification with SDS.P

As promised, we have up to this point totally avoided any examples having to do with `SDS.P`, suggesting that it really is not intended for most programmers and not worth attempting to cover in a high-level introduction such as this one. We also promised one simple sample to show you what it looks like. This sample is also relevant because it is something you cannot do in ADSI at all.

The scenario is that our code needs to connect to an LDAP directory—possibly AD DS or LDS, but also maybe a non-Microsoft LDAP directory—and we need to use SSL.

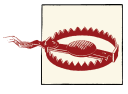
Unfortunately, there is a problem with the server's SSL certificate. Perhaps the subject name on the certificate does not match the DNS name we must use to access the server, or perhaps we do not trust the certificate's issuer on our client, or the certificate has expired. ADSI *does* support SSL/LDAP operations, but it will fail with a "Server Not Operational" error if there is anything wrong with the server's certificate. This behavior cannot be changed, so with ADSI we are out of luck.

However, the Windows LDAP API *does* provide more options here (ADSI simply does not expose access to this advanced feature). Since `SDS.P` provides nearly the entire scope of Windows LDAP functionality, we also have access to this feature in `SDS.P` and can override the SSL verification logic:

```
Public Module MyModule
    Sub Main()
        Dim con As New LdapConnection( _
            new LdapDirectoryIdentifier( _
                "badserver.com:636", True, False))
        con.SessionOptions.SecureSocketLayer = True
        con.SessionOptions.VerifyServerCertificate = _
            AddressOf ServerCallback
        con.Credential = New NetworkCredential("", "")
        con.SessionOptions.SecureSocketLayer = True
        con.AuthType = AuthType.Anonymous
        con.Bind()

        'do a RootDSE search and get the currentTime attribute
        Dim search As New SearchRequest( _
            "", _
            "(objectClass=*)", _
            SearchScope.Base, _
            New String(){"currentTime"} _
            )
        Dim response As SearchResponse = _
            DirectCast(con.SendRequest(search), SearchResponse)
        For Each entry As SearchResultEntry In response.Entries
            Console.WriteLine(entry.Attributes("currentTime")(0))
        Next
    End Sub

    Function ServerCallback( _
        ByVal connection As LdapConnection, _
        ByVal certificate As X509Certificate _
        ) As Boolean
    'ignore errors; do not even check the certificate
    'just return true
        Return True
    End Function
End Module
```

Even though we have tried to scare you with the complexity of `SDS.P`, the code turns out to be fairly simple.

The main trick here is that we define a method called `ServerCallback` that implements a method signature of a specific delegate (basically a .NET callback function) defined in `SDS.P` called `VerifyServerCertificateCallback`. We tell our `LdapConnection` to call our callback function during SSL certificate verification by using the `AddressOf` keyword on the `SessionOptions.VerifyServerCertificate` member to point to our method. `ServerCallback` simply returns `True`, instructing the underlying code to accept the certificate presented by the server and ignore any errors. We could write something more intelligent by looking at the data in the certificate that is passed to us if we wished.



We are not recommending that you ignore SSL errors as a general practice. This is usually a bad idea, especially in cases where the infrastructure is not something you directly control. SSL is there to help protect us. It is better to try to fix the problem with the server's certificate instead, but using the correct hostname or adding required trusted root certificates.

The rest of the code just sets up the connection to use SSL and anonymous authentication and connects to the directory using the `Bind` method. After that, we show a simple `RootDSE` search, a base scope query with a null search base specified, and return the `currentTime` attribute to demonstrate a simple search operation. However, the point here is not to demonstrate the search, but to show the SSL verification. We would not even get past the `Bind` method without our SSL verification override if there was a problem with the server's certificate.

Summary

.NET, now in version 4.5, is here to stay, and Microsoft continues to invest in these resources for developers while the options in native code APIs have been largely unchanged for years now. Not only does .NET allow us to do the same things we are used to doing in our other tools, but it also provides access to features that were previously only available to C++ developers and new approaches that make development easier.

.NET is also an essential tool to learn for tackling other new technologies, such as ASP.NET web development and Windows PowerShell.

In this appendix, we took a quick tour of the now-expansive landscape of the directory services development namespace in .NET, including the following:

- `System.DirectoryServices`
- `System.DirectoryServices.ActiveDirectory`
- `System.DirectoryServices.AccountManagement`
- `System.DirectoryServices.Protocols`

We've only scratched the surface of what we can do, but hopefully we have given you enough to get your feet wet and generate some excitement about the possibilities.

Index

Symbols

A

We'd like to hear your suggestions for improving our indexes. Send email to index@oreilly.com.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

About the Authors

Brian Desmond is a consultant focused on Active Directory, Identity Management, and Identity Federation projects for higher education and commercial enterprise customers. He has worked in numerous large-scale enterprise deployments at various Fortune 100 and larger-scale organizations as well as dozens of K-12 and Higher Education institutions and public sector customers across state and local government.

Since March 2003, Brian has been recognized as a Microsoft MVP for Active Directory for contributions to the Microsoft technical communities at large. He is a frequent contributor to various industry leading publications and a frequent speaker at conferences and events around the world.

Joe Richards is a consultant/admin/tool writer who happens to have a secret identity as a Microsoft MVP for Windows Server Directory Services. His specialty is Directory Services, but he has "minors" in Security and Active Directory programming. He takes time to chat with people on listservs and newsgroups, write about stuff he knows, and whip up various fairly useful tools.

Robbie Allen is a technical leader at Cisco Systems, where he has been involved in the deployment of Active Directory, DNS, DHCP, and several network management solutions. Robbie was named a Windows Server MVP in 2004 and 2005 for his contributions to the Windows community and the publication of several popular O'Reilly books. For more information, see Robbie's website at *http://www.rallenhome.com*.

Alistair G. Lowe-Norris is an Architectural Enterprise Strategy Consultant for Microsoft UK. He worked for Leicester University as the project manager and technical lead of the Rapid Deployment Program for Windows 2000, responsible for rolling out one of the world's largest deployments of Windows 2000 preceding release of the final product.

Colophon

The animals on the cover of *Active Directory*, Fifth 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 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 5 to 30 pounds, with an average of 12 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 80 different colors and patterns. They often live to be 15-20 years old; 10 years for a human life is about equal to 60 years for a cat.

The cat's gestation period is approximately two months, and each litter may contain three to seven cats. Mother cats teach their kittens to eat and to use litter boxes. Kittens

ideally should not leave their mother's sides until the age of 12 weeks and are considered full grown at the age of about three years.

The cover image is a 19th-century engraving from Dover Pictorial Archive. The cover font is Adobe ITC Garamond. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.