Microsoft

Inside OUT

The ultimate, in-depth reference

Hundreds of timesaving solutions

Supremely organized, packed with expert advice

Windows Server 2012 R2: Services, Security, & Infrastructure

William R. Stanek *Windows technologies expert + award-winning author*

Microsoft

Windows Server 2012 R2 Inside Out: Services, Security, & Infrastructure

William R. Stanek

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2013955708
ISBN: 978-0-7356-8255-9

Printed and bound in the United States of America.

First Printing: April 2014

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at http://aka.ms/tellpress.

Acquisitions Editor: Anne Hamilton
Developmental Editor: Karen Szall
Project Editor: Rosemary Caperton
Editorial Production: nSight, Inc.
Technical Reviewer: Charlie Russel; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Copyeditor: Joseph Gustaitis
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

To my readers—Windows Server 2012 R2 Inside Out: Services, Security, & Infrastructure *is my 49th book for Microsoft Press. Thank you for being there with me through many books and many years. It's been an honor and a privilege.*

To my wife—for many years, through many books, many millions of words, and many thousands of pages she's been there, providing support and encouragement and making every place we've lived a home.

To my kids—for helping me see the world in new ways, for having exceptional patience and boundless love, and for making every day an adventure.

To Anne, Karen, Martin, Lucinda, Juliana, and many others who've helped out in ways both large and small.

Special thanks to my son Will for not only installing and managing my extensive dev lab for all my books since Windows 8 Pocket Consultant *but for also performing check reads of all those books as well.*

—William R. Stanek

Contents at a glance

Table of contents

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can improve our books and learning resources for you. To participate in a brief survey, please visit:

http://aka.ms/tellpress

Introduction

Welcome to *Windows Server 2012 R2 Inside Out: Services, Security, & Infrastructure*. As the author of many popular technology books, I've been writing professionally about Microsoft Windows and Windows Server since 1994. Over the years I've gained a unique perspective—the kind of perspective you can gain only after working with technologies for a long time. The advantage for you, the reader, is that my solid understanding of these technologies allowed me to dig into Windows Server 2012 R2 architecture, internals, and configuration to see how things really work under the hood and then pass this information on to you throughout this book.

Anyone transitioning to Windows Server 2012 R2 from Windows Server 2012 might be surprised at just how much has been updated as changes both subtle and substantial have been made throughout the operating system. For anyone transitioning to Windows Server 2012 R2 from Windows Server 2008 R2 or an earlier release of Windows Server, I'll let you know right up front that Windows Server 2012 and Windows Server 2012 R2 are substantially different from earlier versions of Windows Server. Not only are there major changes throughout the operating system, but also this just might be the first version of Windows Server that you manage using a touch-based user interface. If you do end up managing it this way, mastering the touch-based UI and the revised interface options will be essential for your success. For this reason, I discuss both the touch UI and the traditional mouse and keyboard techniques throughout this book.

When you are working with touch UI–enabled computers, you can manipulate onscreen elements in ways that weren't previously possible. You can enter text using the onscreen keyboard and manipulate onscreen elements in the following ways:

- **Tap.** Tap an item by touching it with your finger. A tap or double-tap of elements on the screen is generally the equivalent of a mouse click or double-click.
- **Press and hold.** Press your finger down and leave it there for a few seconds. Pressing and holding elements on the screen is generally the equivalent of a right-click.
- **Swipe to select.** Slide an item a short distance in the opposite direction from how the page scrolls. This selects the items and also might bring up related commands. If pressing and holding doesn't display commands and options for an item, try swiping to select instead.
- **Swipe from edge (slide in from edge).** Starting from the edge of the screen, swipe or slide in. Sliding in from the right edge opens the Charms panel. Sliding in from the left edge shows open apps and allows you to easily switch between

them. Sliding in from the top or bottom edge shows commands for the active element.

- **Pinch.** Touch an item with two or more fingers and then move those fingers toward each other. Pinching zooms out.
- **Stretch.** Touch an item with two or more fingers and then move those fingers away from each other. Stretching zooms in.

In this book I teach you how server roles, role services, and features work; why they work the way they do; and how to customize them to meet your needs. Regardless of your job title, if you're deploying, configuring, managing, or maintaining Windows Server 2012 R2, this book is for you. To pack in as much information as possible, I had to assume that you have basic networking skills and a basic understanding of Windows Server and that you are familiar with Windows commands and procedures. With this in mind, I don't devote entire chapters to basic skills or to why you want to use Windows Server. Instead, I focus on essential services, infrastructure servers, and security.

Conventions

The following conventions are used in this book:

- **Abbreviated menu commands.** For your convenience, this book uses abbreviated menu commands. For example, "Tap or click Tools, Track Changes, Highlight Changes" means that you should tap or click the Tools menu, select Track Changes, and then tap or click the Highlight Changes command.
- **Boldface type.** **Boldface** type is used to indicate text that you enter or type.
- **Initial Capital Letters.** The first letters of the names of menus, dialog boxes, dialog box elements, and commands are capitalized. Example: the Save As dialog box.
- ***Italicized type.*** *Italicized* type is used to indicate new terms.
- **Plus sign (+) in text.** Keyboard shortcuts are indicated by a plus sign (+) separating two key names. For example, Ctrl+Alt+Delete means that you press the Ctrl, Alt, and Delete keys at the same time.

How to reach the author

Email: williamstanek@aol.com

Web: *http://www.williamrstanek.com/*

Facebook: *https://www.facebook.com/William.Stanek.Author*

Twitter: *http://twitter.com/williamstanek*

Errata and book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

http://aka.ms/WSIO_R2_SSI

If you discover an error that is not already listed, please submit it to us at the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

We want to hear from you

At Microsoft Press your satisfaction is our top priority and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

CHAPTER 10

Active Directory architecture

Active Directory is an extensible directory service that enables you to manage network resources efficiently. A directory service does this by storing detailed information about each network resource, which makes it easier to provide basic lookup and authentication. Being able to store large amounts of information is a key objective of a directory service, but the information must also be organized so that it's easily searched and retrieved.

Active Directory provides for authenticated search and retrieval of information by dividing the physical and logical structures of the directory into separate layers. Understanding the physical structure of Active Directory is important for understanding how a directory service works. Understanding the logical structure of Active Directory is important for implementing and managing a directory service.

Active Directory physical architecture

The physical layer of Active Directory controls the following features:

- How directory information is accessed
- How directory information is stored on the hard disk of a server

Active Directory physical architecture: A top-level view

From a physical or machine perspective, Active Directory is part of the security subsystem. (See Figure 10-1.) The security subsystem runs in user mode. User-mode applications do not have direct access to the operating system or hardware. This means that requests from user-mode applications have to pass through the executive services layer and must be validated before being executed.

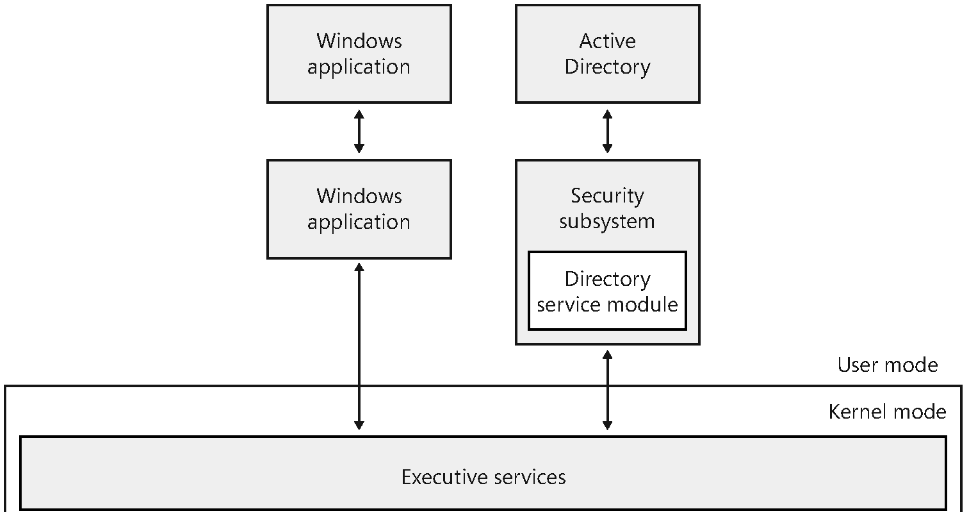

Figure 10-1 Top-level overview of the Active Directory architecture.

NOTE

Being part of the security subsystem makes Active Directory an integrated part of the access-control and authentication mechanism built into Microsoft Windows Server. Access control and authentication protect the resources in the directory.

Each resource in Active Directory is represented as an object. Anyone who tries to gain access to an object must be granted permission. Lists of permissions that describe who or what can access an object are referred to as *access control lists (ACLs)*. Each object in the directory has an associated ACL.

You can restrict permissions across a broader scope by using Group Policy. The security infrastructure of Active Directory uses policy to enforce security models on several objects that are grouped logically. You can also set up trust relationships between groups of objects to allow for an even broader scope for security controls between trusted groups of objects that need to interact. From a top-level perspective, that's how Active Directory works, but to really understand Active Directory, you need to delve into the security subsystem.

Active Directory within the Local Security Authority

Within the security subsystem, Active Directory is a subcomponent of the Local Security Authority (LSA). As shown in Figure 10-2, the LSA consists of many components that provide the security features of Windows Server and ensure that access control and authentication function as they should. Not only does the LSA manage local security policy but it also performs the following functions:

- Generates security identifiers (SIDs)
- Provides the interactive process for logon
- Manages auditing

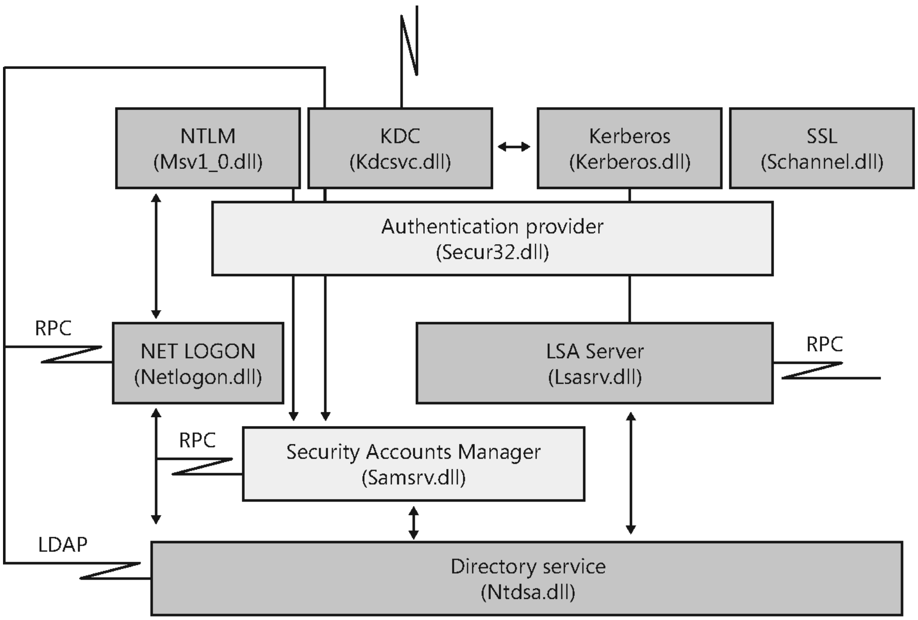

Figure 10-2 Windows Server security subsystem using Active Directory.

When you work through the security subsystem as it is used with Active Directory, you'll find the three following key areas:

- Authentication mechanisms
  - **NTLM (Msv1_0.dll).** Used for Windows NT LAN Manager (NTLM) authentication
  - **Kerberos (Kerberos.dll) and Key Distribution Center (Kdcsvc.dll).** Used for Kerberos V5 authentication
  - **SSL (Schannel.dll).** Used for Secure Sockets Layer (SSL) authentication
  - **Authentication provider (Secur32.dll).** Used to manage authentication

- Logon/access-control mechanisms
    - **NET LOGON (Netlogon.dll).** Used for interactive logon through NTLM. For NTLM authentication, NET LOGON passes logon credentials to the directory service module and returns the SIDs for objects to clients making requests.
    - **LSA Server (Lsasrv.dll).** Used to enforce security policies for Kerberos and SSL. For Kerberos and SSL authentication, LSA Server passes logon credentials to the directory service module and returns the SIDs for objects to clients making requests.
    - **Security Accounts Manager (Samsrv.dll).** Used to enforce security policies for NTLM.
- **Directory service component: Directory service (Ntdsa.dll).** Used to provide directory services for Windows Server. This is the actual module that allows you to perform authenticated searches and retrieval of information.

As you can see, users are authenticated before they can work with the directory service component. Authentication is handled by passing a user's security credentials to a domain controller. After the user is authenticated on the network, the user can work with resources and perform actions according to the permissions and rights the user has been granted in the directory. At least, this is how the Windows Server security subsystem works with Active Directory.

When you are on a network that doesn't use Active Directory, or when you log on locally to a machine other than a domain controller, the security subsystem works as shown in Figure 10-3. Here, the directory service is not used. Instead, authentication and access control are handled through the Security Accounts Manager (SAM). Here, information about resources is stored in the SAM, which itself is stored in the registry.

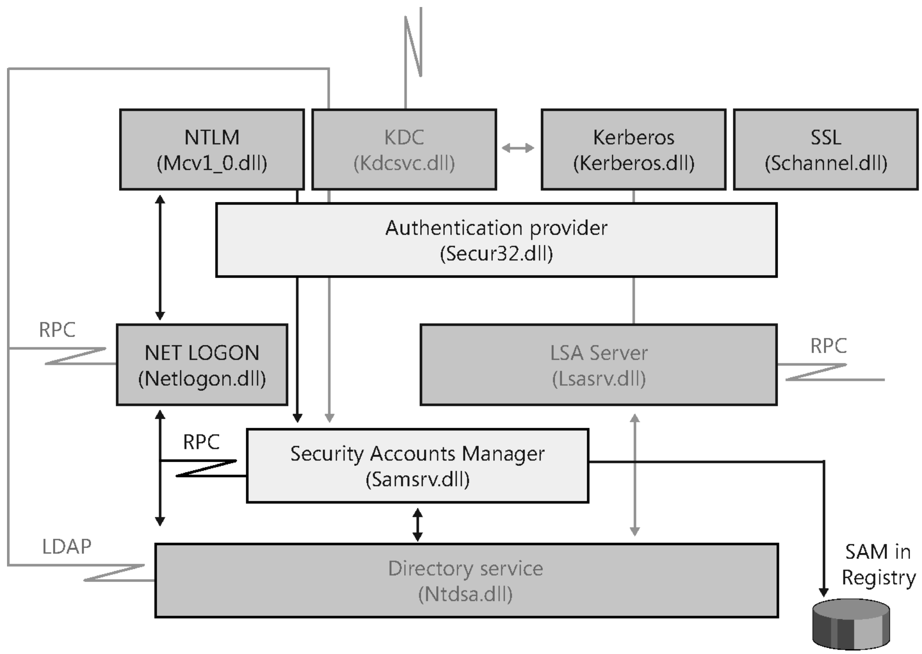

Figure 10-3 Windows Server security subsystem without Active Directory.

Directory service architecture

As you've seen, incoming requests are passed through the security subsystem to the directory service component. The directory service component is designed to accept requests from many kinds of clients. As shown in Figure 10-4, these clients use specific protocols to interact with Active Directory.

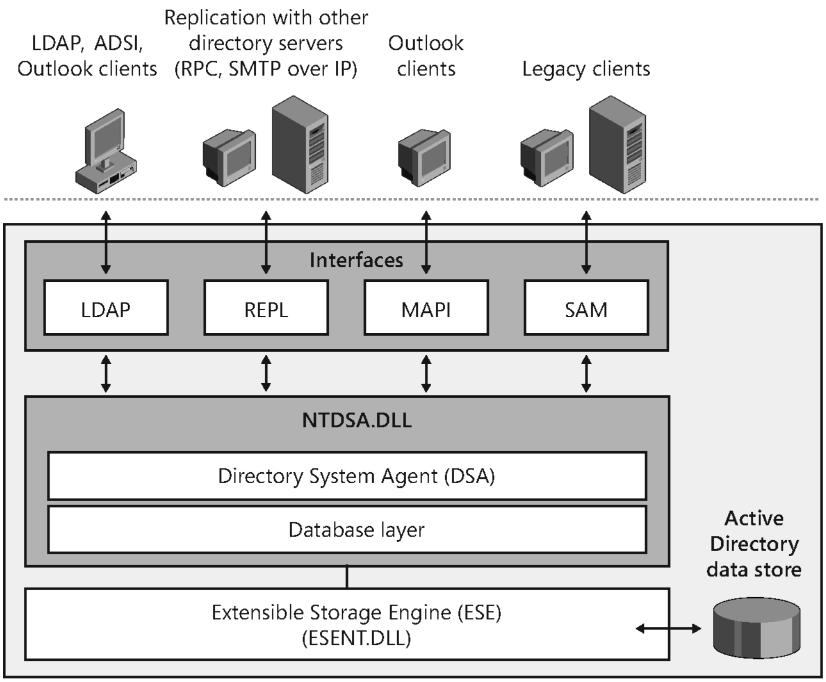

Figure 10-4 The directory service architecture.

Protocols and client interfaces

The primary protocol for Active Directory access is Lightweight Directory Access Protocol (LDAP). LDAP is an industry standard protocol for directory access that runs over Transmission Control Protocol/Internet Protocol (TCP/IP). Active Directory supports LDAP versions 2 and 3. Clients can use LDAP to query and manage directory information—depending on the level of permissions they have been granted—by establishing a TCP connection to a domain controller. The default TCP port used by LDAP clients is 389 for standard communications and 636 for SSL.

Active Directory supports intersite and intrasite replication through the REPL interface, which uses either remote procedure calls (RPCs) or Simple Mail Transfer Protocol over Internet Protocol (SMTP over IP), depending on how replication is configured. Each domain controller is responsible for replicating changes to the directory to other domain controllers, using a multimaster approach. The multimaster approach used in Active Directory allows updates to be made to the directory by any domain controller and then replicated to other domain controllers.

For older messaging clients, Active Directory supports the Messaging Application Programming Interface (MAPI). MAPI allows messaging clients to access Active Directory (which Microsoft Exchange uses for storing information), primarily for address book lookups. Messaging clients use RPCs to establish a connection with the directory service. The RPC Endpoint Mapper uses UDP port 135 and TCP port 135. Current messaging clients use LDAP instead of RPC.

For legacy clients, Active Directory supports the SAM interface, which also uses RPCs. This allows legacy clients to access the Active Directory data store the same way they would access the SAM database. The SAM interface is also used during certain replication activities.

Directory System Agent and database layer

Clients and other servers use the LDAP, REPL, MAPI, and SAM interfaces to communicate with the directory service component (Ntdsa.dll) on a domain controller. From an abstract perspective, the directory service component consists of the following:

- Directory System Agent (DSA), which provides the interfaces through which clients and other servers connect
- Database layer, which provides an application programming interface (API) for working with the Active Directory data store

From a physical perspective, the DSA is really the directory service component and the database layer resides within it. The reason for separating the two is that the database layer performs a vital abstraction. Without this abstraction, the physical database on the disk would not be protected from the applications the DSA interacts with. Furthermore, the object-based hierarchy used by Active Directory would not be possible. Why? Because the data store is in a single data file using a flat (record-based) structure, whereas the database layer is used to represent the flat file records as objects within a hierarchy of containers. Like a folder that can contain files and other folders, a container is simply a type of object that can contain other objects and other containers.

Each object in the data store has a name relative to the container in which it's stored. This name is aptly called the object's *relative distinguished name (RDN)*. An object's full name, also referred to as an object's *distinguished name (DN)*, describes the series of containers, from the highest to the lowest, of which the object is a part.

To make sure every object stored in Active Directory is truly unique, each object also has a globally unique identifier (GUID), which is generated when the object is created. Unlike an object's RDN or DN, which can be changed by renaming an object or moving it to another container, the GUID can never be changed. The DSA assigns it to an object, and it never changes.

The DSA is responsible for ensuring that the type of information associated with an object adheres to a specific set of rules. This set of rules is referred to as the *schema*. The schema is stored in the directory and contains the definitions of all object classes and describes their attributes. In Active Directory the schema is the set of rules that determine the kind of data that can be stored in the database, the type of information that can be associated with a particular object, the naming conventions for objects, and so on.

Inside OUT

The schema saves space and helps validate attributes

The schema serves to separate an object's definition from its actual values. Thanks to the schema, Active Directory doesn't have to write information about all of an object's possible attributes when it creates the object. When you create an object, only the defined attributes are stored in the object's record. This saves a lot of space in the database. Furthermore, because the schema specifies not only the valid attributes but also the valid values for those attributes, Active Directory uses the schema both to validate the attributes that have been set on an object and to keep track of what other possible attributes are available.

The DSA is also responsible for enforcing security limitations. It does this by reading the SIDs on a client's access token and comparing them to the SIDs for an object. If a client has appropriate access permissions, it is granted access to an object. If a client doesn't have appropriate access permissions, it's denied access.

Finally, the DSA is used to initiate replication. Replication is the essential functionality that ensures that the information stored on domain controllers is accurate and consistent with changes that have been made. Without proper replication, the data on servers would become stale and outdated.

Extensible Storage Engine

Active Directory uses the Extensible Storage Engine (ESE) to retrieve information from, and write information to, the data store. The ESE uses indexed and sequential storage with transactional processing, as follows:

- **Indexed storage.** Indexing the data store allows the ESE to access data quickly without having to search the entire database. In this way, the ESE can rapidly retrieve, write, and update data.
- **Sequential storage.** Sequentially storing data means that the ESE writes data as a stream of bits and bytes. This allows data to be read from and written to specific locations.

- **Transactional processing.** Transactional processing ensures that changes to the database are applied as discrete operations that can be rolled back if necessary.

Any data that is modified in a transaction is copied to a temporary database file. This gives two views of the data that's being changed: one view for the process changing the data and one view of the original data that's available to other processes until the transaction is finalized. A transaction remains open as long as changes are being processed. If an error occurs during processing, the transaction can be rolled back to return the object being modified to its original state. If Active Directory finishes processing changes without errors occurring, the transaction can be committed.

As with most databases that use transactional processing, Active Directory maintains a transaction log. A record of the transaction is written first to an in-memory copy of an object, then to the transaction log, and finally to the database. The in-memory copy of an object is stored in the *version store*. The version store is an area of physical memory (RAM) used for processing changes. Typically, the version store is 25 percent of the physical RAM.

The transaction log serves as a record of all changes that have yet to be committed to the database file. The transaction is written first to the transaction log to ensure that even if the database shuts down immediately afterward, the change is not lost and can take effect. To ensure this, Active Directory uses a checkpoint file to track the point up to which transactions in the log file have been committed to the database file. After a transaction is committed to the database file, it can be cleared out of the transaction log.

The actual update of the database is written from the in-memory copy of the object in the version store and not from the transaction log. This reduces the number of disk I/O operations and helps ensure that updates can keep pace with changes. When many updates are made, however, the version store can reach a point at which it's overwhelmed. This happens when the version store reaches 90 percent of its maximum size. When this happens, the ESE temporarily stops processing cleanup operations that are used to return space after an object is modified or deleted from the database.

Although in earlier releases of Windows Server index creation could affect domain controller performance, Windows Server 2012 and Windows Server 2012 R2 allow you to defer index creation to a time when it's more convenient. By deferring index creation to a designated point in time, rather than creating indexes as needed, you can ensure that domain controllers can perform related tasks during off-peak hours, thereby reducing the impact of index creation. Any attribute that is in a deferred index state will be logged in the event log every 24 hours. Look for event IDs 2944 and 2945. When indexes are created, event ID 1137 is logged.

In large Active Directory environments, deferring index creation is useful to prevent domain controllers from becoming unavailable due to building indexes after schema updates. Before you can use deferred index creation, you must enable the feature in the forest root domain. You do this using the *DSHeuristics* attribute of the Directory Services object for the domain. Set the eighteenth bit of this attribute to 1. Because the tenth bit of this attribute typically also is set to 1 (if the attribute is set to a value), the attribute normally is set to the following: 000000000100000001. You can modify the *DSHeuristics* attribute using ADSI Edit or Ldp.exe.

ADSI Edit is a snap-in you can add to any Microsoft Management Console (MMC). Open a new MMC by entering **MMC** at a prompt and then use the Add/Remove Snap-in option on the File menu to add the ADSI Edit snap-in to the MMC. You can then use ADSI Edit to modify the *DSHeuristics* attribute by completing the following steps:

1. Press and hold or right-click the root node and then select Connect To. In the Connection Settings dialog box, choose the Select A Well Known Naming Context option. On the related selection list, select Configuration (because you want to connect to the Configuration naming context for the domain) and then tap or click OK.
2. In ADSI Edit, work your way down to the CN=Directory Service container by expanding the Configuration naming context, the CN=Configuration container, the CN=Services container, and the CN=Windows NT container.
3. Next, press and hold or right-click CN=Directory Service and then select Properties. In the Properties dialog box, select the dsHeuristics properties and then tap or click Edit.
4. In the String Attribute Editor dialog box, type the desired value, such as **000000000100000001**, and then tap or click OK twice.

Ldp is a graphical utility. Open Ldp by typing **ldp** in the Apps Search box or at a prompt. You can then use Ldp to modify the *DSHeuristics* attribute by completing the following steps:

1. Choose Connect on the Connection menu and then connect to a domain controller in the forest root domain. After you connect to a domain controller, choose Bind on the Connection menu to bind to the forest root domain using an account with enterprise administrator privileges.
2. Next, choose Tree on the View menu to open the Tree View dialog box. In the Tree View dialog box, choose CN=Configuration container as the base distinguished name to work with.
3. In the CN=Configuration container, expand the CN=Services container, expand the CN=Windows NT container, and then select the CN=Directory Service container. Next, press and hold or right-click CN=Directory Service and then select Modify.
4. In the Modify dialog box, type the attribute name as **dsHeuristics** and the value as **000000000100000001**.

5. If the attribute already exists, set the Operation as Replace. Otherwise, set the Operation as Add.

6. Tap or click Enter to create an LDAP transaction for this update, and then tap or click Run to apply the change.

NOTE

The value 000000000100000001 is nine zeros with a 1 in the tenth position followed by seven zeros with a 1 in the eighteenth position.

Once the change is replicated to all domain controllers in the forest, they will defer index creation automatically. You must then trigger index creation manually by either restarting domain controllers, which rebuilds the schema cache and deferred indexes, or by triggering a schema update for the RootDSE. In ADSI Edit, you can initiate an update by connecting to the RootDSE. To do this, press and hold or right-click the root node and then select Connect To. In the Connection Settings dialog box, choose the Select A Well Known Naming Context option. On the related selection list, select RootDSE and then tap or click OK. In ADSI Edit, press and hold or right-click the RootDSE node and then select Update Schema Now.

To allow for object recovery and for the replication of object deletions, an object that is deleted from the database is logically removed rather than physically deleted. The way deletion works depends on whether Active Directory Recycle Bin is enabled or disabled.

Deletion without Recycle Bin When Active Directory Recycle Bin is disabled, as with standard deployments prior to Windows Server 2008 R2, most of the object's attributes are removed and the object's *Deleted* attribute is set to TRUE to indicate that it has been deleted. The object is then moved to a hidden Deleted Objects container where its deletion can be replicated to other domain controllers. (See Figure 10-5.) In this state, the object is said to be *tombstoned*. To allow the tombstoned state to be replicated to all domain controllers, and thus removed from all copies of the database, an attribute called *tombstoneLifetime* is also set on the object. The *tombstoneLifetime* attribute specifies how long the tombstoned object should remain in the Deleted Objects container. The default lifetime is 180 days.

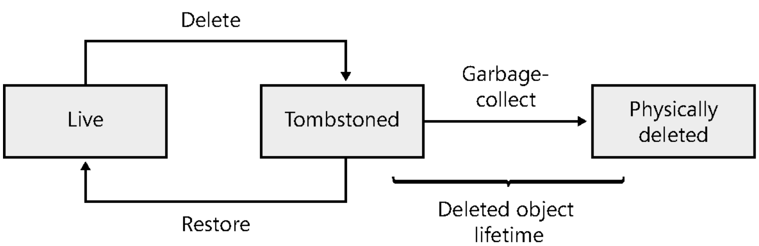

Figure 10-5 Active Directory object life cycle without Recycle Bin.

Inside OUT

The tombstone process

When an object is tombstoned, Active Directory changes the distinguished name so that the object name can't be recognized. Next, Active Directory deletes all of the object's link-valued attributes, and most of the object's non-link-valued attributes are cleared. Finally, the object is moved to the Deleted Objects container.

You can recover tombstoned objects using tombstone reanimation. However, attribute values that were removed are not recovered. This means the link-valued attributes, which include group memberships of user accounts, and the non-link-valued attributes are not recovered.

The ESE uses a garbage-collection process to clear out tombstoned objects after the tombstone lifetime has expired, and it performs automatic online defragmentation of the database after garbage collection. The interval at which garbage collection occurs is a factor of the value set for the *garbageCollPeriod* attribute and the tombstone lifetime. By default, garbage collection occurs every 12 hours. When there are more than 5,000 tombstoned objects to be garbage-collected, the ESE removes the first 5,000 tombstoned objects and then uses the CPU availability to determine if garbage collection can continue. If no other process is waiting for the CPU, garbage collection continues for up to the next 5,000 tombstoned objects whose tombstone lifetime has expired, and the CPU availability is again checked to determine if garbage collection can continue. This process continues until all the tombstoned objects whose tombstone lifetime has expired are deleted or another process needs access to the CPU.

Deletion with Recycle Bin When Active Directory Recycle Bin is enabled as an option with Windows Server 2008 R2 and later, objects aren't tombstoned when they are initially deleted and their attributes aren't removed. Instead, the deletion process occurs in stages.

In the first stage of the deletion, the object is said to be *logically deleted*. Here, the object's *Deleted* attribute is set to TRUE to indicate that it has been deleted. The object is then moved, with its attributes and name preserved, to a hidden Deleted Objects container where its deletion can be replicated to other domain controllers. (See Figure 10-6.) To allow the logically deleted state to be replicated to all domain controllers, and thus removed from all copies of the database, an attribute called *ms-DeletedObjectLifetime* is also set on the object. The *ms-DeletedObjectLifetime* attribute specifies how long the logically deleted object should remain in the Deleted Objects container. The default deleted object lifetime is 180 days.

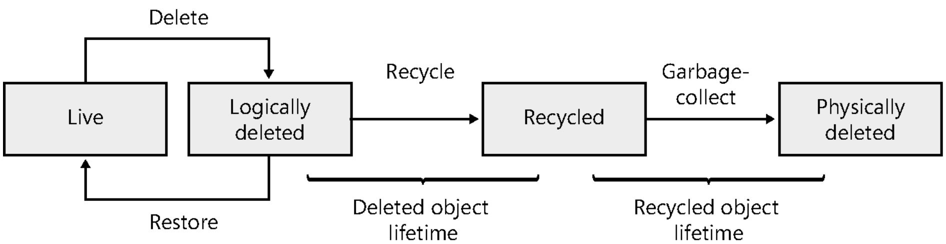

Figure 10-6 Active Directory object life cycle with Recycle Bin.

When the deleted object lifetime expires, Active Directory removes most of the object's attributes, changes the distinguished name so that the object name can't be recognized, and sets the object's *tombstoneLifetime* attribute. This effectively tombstones the object (and the process is the same as the legacy tombstone process).

The recycled object remains in the Deleted Objects container until the recycled object lifetime expires, and it's said to be in the *recycled* state. The default tombstone lifetime is 180 days.

As with deletion without the Recycle Bin, the ESE uses a garbage-collection process to clear out tombstoned objects after the tombstone lifetime has expired. This garbage-collection process is the same as discussed previously.

Data store architecture

After you examine the operating system components that support Active Directory, the next step is to see how directory data is stored on a domain controller's hard disks. As Figure 10-7 shows, the data store has a primary data file and several other types of related files, including working files and transaction logs.

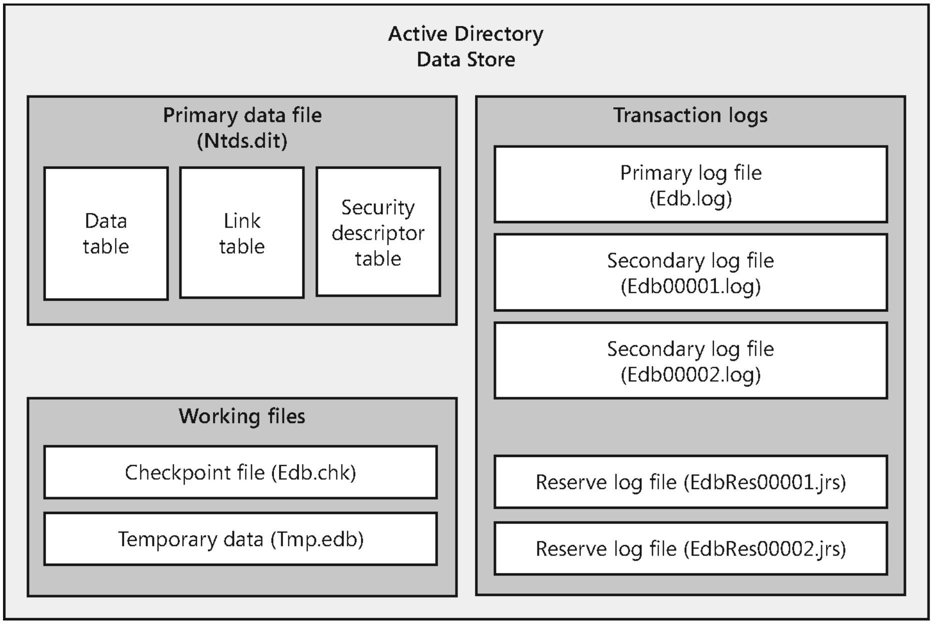

Figure 10-7 The Active Directory data store.

These files are used as follows:

- **Primary data file (Ntds.dit).** Physical database file that holds the contents of the Active Directory data store
- **Checkpoint file (Edb.chk).** Checkpoint file that tracks the point up to which the transactions in the log file have been committed to the database file
- **Temporary data (Tmp.edb).** Temporary workspace for processing transactions
- **Primary log file (Edb.log).** Primary log file that contains a record of all changes that have yet to be committed to the database file
- **Secondary log files (Edb00001.log, Edb00002.log, ...).** Additional logs files that are used as needed
- **Reserve log files (EdbRes00001.jrs, EdbRes00002.jrs, ...).** Files that are used to reserve space for additional log files if the primary log file becomes full

The primary data file contains three indexed tables:

- **Active Directory data table.** The data table contains a record for each object in the data store, which can include object containers, the objects themselves, and any other type of data that is stored in Active Directory.
- **Active Directory link table.** The link table is used to represent linked attributes. A linked attribute is an attribute that refers to other objects in Active Directory. For example, if an object contains other objects (that is, it is a container), attribute links are used to point to the objects in the container.
- **Active Directory security descriptor table.** The security descriptor table contains the inherited security descriptors for each object in the data store. Windows Server uses this table so that inherited security descriptors no longer have to be duplicated on each object. Instead, inherited security descriptors are stored in this table and linked to the appropriate objects. This makes Active Directory authentication and control mechanisms very efficient.

Think of the data table as having rows and columns; the intersection of a row and a column is a *field*. The table's rows correspond to individual instances of an object. The table's columns correspond to attributes defined in the schema. The table's fields are populated only if an attribute contains a value. Fields can be a fixed or a variable length. If you create an object and define only 10 attributes, only these 10 attributes will contain values. Although some of those values might be fixed length, others might be variable length.

Records in the data table are stored in data pages that have a fixed size of 8 kilobytes (KBs, or 8,192 bytes). Each data page has a page header, data rows, and free space that can contain row offsets. The page header uses the first 96 bytes of each page, leaving 8,096 bytes for data and row offsets.

Row offsets indicate the logical order of rows on a page, which means that offset 0 refers to the first row in the index, offset 1 refers to the second row, and so on. If a row contains long, variable-length data, the data might not be stored with the rest of the data for that row. Instead, Active Directory can store an 8-byte pointer to the actual data, which is stored in a collection of 8 KB pages that aren't necessarily written contiguously. In this way, an object and all its attribute values can be much larger than 8 KBs.

The primary log file has a fixed size of 10 megabytes (MBs). When this log fills up, Active Directory creates additional (secondary) log files as necessary. The secondary log files are also limited to a fixed size of 10 MBs. Active Directory uses the reserve log files to reserve space on disk for log files that might need to be created. Because several reserve files are already created, this speeds up the transactional logging process when additional logs are needed.

By default, the primary data file, the working files, and the transaction logs are all stored in the same location. On a domain controller's system volume, you'll find these files in the %SystemRoot%\NTDS folder. Although these are the only files used for the data store, Active Directory uses other files. For example, policy files and other files, such as startup and shutdown scripts used by the DSA, are stored in the %SystemRoot%\Sysvol folder.

NOTE

A distribution copy of Ntds.dit is also placed in the %SystemRoot%\System32 folder. This is used to create a domain controller when you install Active Directory on a server running Windows Server. If the file doesn't exist, the Active Directory Installation Wizard will need the installation media to promote a member server to be a domain controller.

Inside OUT

The log files have attributes you can examine

When you stop Active Directory Domain Services, you can use the Extensible Storage Engine Utility (esentutl.exe) to examine log file properties. At an elevated command prompt, type esentutl.exe –ml LogName, where *LogName* is the name of the log file to examine, such as edb.log, to obtain detailed information on the log file, including the base name, creation time, format version, log sector sizes, and logging parameters. While Active Directory Domain Services is offline, you can also use esentutl.exe to perform defragmentation, integrity checks, and copy, repair, and recovery operations. To learn more about this utility, type esentutl.exe at an elevated command prompt. Following the prompts, you can then type the letter corresponding to the operation you want to learn more about. For example, type esentutl.exe and then press the D key to learn the defragmentation options.

Active Directory logical architecture

The logical layer of Active Directory determines how you see the information contained in the data store and also controls access to that information. The logical layer does this by defining the namespaces and naming schemes used to access resources stored in the directory. This provides a consistent way to access directory-stored information regardless of type. For example, you can obtain information about a printer resource stored in the directory in much the same way that you can obtain information about a user resource.

To better understand the logical architecture of Active Directory, you need to understand the following topics:

- Active Directory objects
- Active Directory domains, trees, and forests
- Active Directory trusts
- Active Directory namespaces and partitions
- Active Directory data distribution

Active Directory objects

Because so many types of resources can be stored in the directory, a standard storage mechanism was needed and Microsoft developers decided to use the LDAP model for organizing data. In this model, each resource that you want to represent in the directory is created as an object with attributes that define information you want to store about the resource. For example, the user object in Active Directory has attributes for a user's first name, middle initial, last name, and logon name.

An object that holds other objects is referred to as a *container object* or simply a *container*. The data store itself is a container that contains other containers and objects. An object that can't contain other objects is a *leaf object*. Each object created within the directory is of a particular type or class. The object classes are defined in the schema. Some of the object types include:

- User
- Group
- Computer
- Printer

When you create an object in the directory, you must comply with the schema rules for that object class. Not only do the schema rules dictate the available attributes for an object class, they also dictate which attributes are mandatory and which attributes are optional. When you create an object, mandatory attributes must be defined. For example, you can't create a user object without specifying the user's full name and logon name. The reason is that these attributes are mandatory.

Some rules for attributes also are defined in policy. For example, the default security policy for Windows Server specifies that a user account must have a password and that the password must meet certain complexity requirements. If you try to create a user account without a password or with a password that doesn't meet these complexity requirements, the account creation will fail because of the security policy.

The schema also can be extended or changed. This allows administrators to define new object classes, add attributes to existing objects, and change the way attributes are used. However, you need special access permissions and privileges to work directly with the schema. Specifically, you must be a member of the Schema Admins group.

Active Directory domains, trees, and forests

Within the directory, objects are organized using a hierarchical tree structure called a *directory tree*. The structure of the hierarchy is derived from the schema and is used to define the parent–child relationships of objects stored in the directory.

A logical grouping of objects that allows central management of those objects is called a *domain*. In the directory tree, a domain is itself represented as an object. In fact, it's the parent object of all the objects it contains. An Active Directory domain can contain millions of objects. You can create a single domain that contains all the resources you want to manage centrally. In Figure 10-8, a domain object is represented by a large triangle and the objects it contains are as shown.

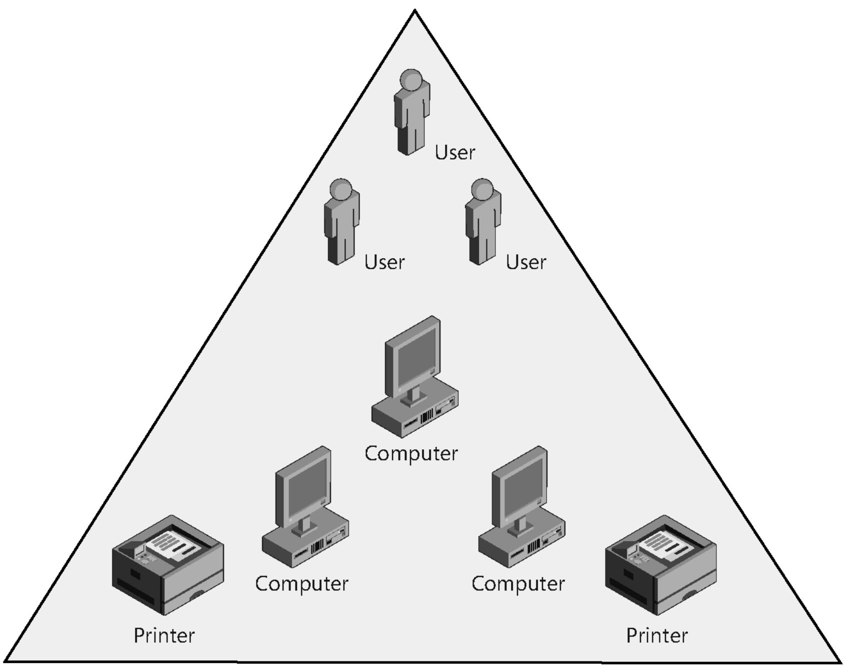

Figure 10-8 An Active Directory domain.

Domains are only one of several building blocks for implementing Active Directory structures. Other building blocks include the following:

- Active Directory trees, which are logical groupings of domains
- Active Directory forests, which are logical groupings of domain trees

As described, a directory tree is used to represent a hierarchy of objects, showing the parent–child relationships between those objects. Thus, when we're talking about a domain tree, we're looking at the relationship between parent and child domains. The domain at the top of the domain tree is referred to as the *root domain* (think of this as an upside-down tree). More specifically, the root domain is the first domain created in a new tree within Active Directory. When talking about forests and domains, there is an important distinction made between the first domain created in a new forest—a forest root domain—and the first domain created in each additional tree within a forest—a root domain.

In the example shown in Figure 10-9, cohovineyard.com is the root domain in an Active Directory forest with a single tree—that is, it's the forest root domain. As such, cohovineyard.com is the parent of the sales.cohovineyard.com domain and the mf.cohovineyard.com domain. The mf.cohovineyard.com domain itself has a related subdomain: bottling.mf.cohovineyard.com. This makes mf.cohovineyard.com the parent of the child domain bottling.mf.cohovineyard.com.

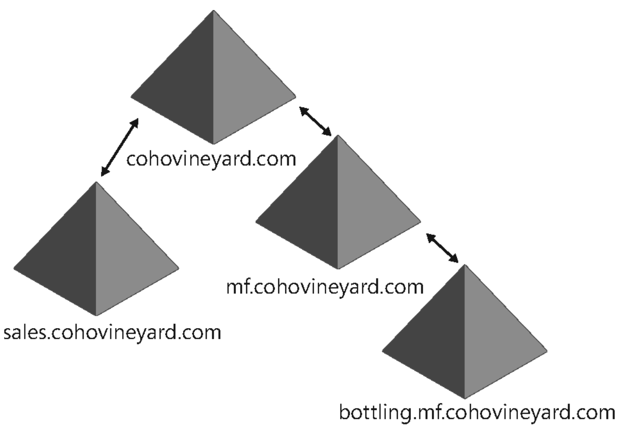

Figure 10-9 An Active Directory forest with a single tree.

The most important thing to note about this and all domain trees is that the namespace is contiguous. Here, all the domains are part of the cohovineyard.com namespace. If a domain is a part of a different namespace, it can be added as part of a new tree in the forest. In the example shown in Figure 10-10, a second tree is added to the forest. The root domain of the second tree is cohowinery.com, and this domain has cs.cohowinery.com as a child domain.

CHAPTER 10

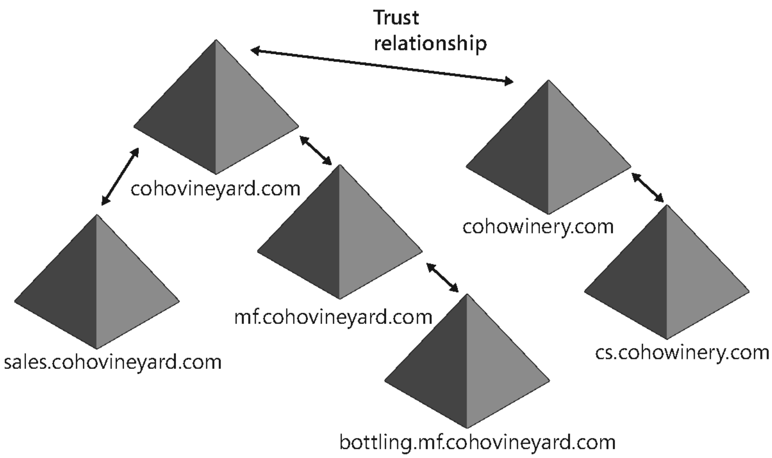

Figure 10-10 An Active Directory forest with multiple trees.

You create a forest root domain by installing Active Directory on a stand-alone server and establishing the server as the first domain controller in a new forest. To add a tree to an existing forest, you install Active Directory on a stand-alone server and configure the server as a member of the forest, but with a domain name that is not part of the current namespace being used. You make the new domain part of the same forest to allow associations called *trusts* to be made between domains that belong to different namespaces.

Active Directory trusts

In Active Directory, two-way transitive trusts are established automatically between domains that are members of the same forest. Trusts join parent and child domains in the same domain tree and join the roots of domain trees. Trusts are transitive, which means that if domain A trusts domain B and domain B trusts domain C, domain A trusts domain C. Because all trusts in Active Directory are two-way and transitive, by default every domain in a forest implicitly trusts every other domain. It also means that resources in any domain are available to users in every domain in the forest. For example, with the trust relationships in place, a user in the sales.cohovineyard.com domain could access a printer or other resources in the cohovineyard.com domain—or even the cs.cohowinery.com domain.

However, the creation of a trust doesn't imply any specific permission. Instead, it implies only the ability to grant permissions. No privileges are automatically implied or inherited by the establishment of a trust relationship. The trust doesn't grant or deny any permission. It exists only to allow administrators to grant permissions.

Several key terms are used to describe trusts, including the following:

- **Trusting domain.** A domain that establishes a trust is referred to as a *trusting domain*. Trusting domains allow access by users from another domain (the trusted domain).
- **Trusted domain.** A domain that trusts another domain is referred to as a *trusted domain*. Users in trusted domains have access to another domain (the trusting domain).

To make it easier for administrators to grant access throughout a forest, Active Directory allows you to designate two types of administrators:

- **Enterprise administrators.** These are the designated administrators of the enterprise. Enterprise administrators can manage and grant access to resources in any domain in the Active Directory forest.
- **Domain administrators.** These are the designated administrators of a particular domain. Domain administrators in a trusting domain can access user accounts in a trusted domain and set permissions that grant access to resources in the trusting domain.

Going back to the example, Tom, an enterprise administrator in this forest, could grant access to resources in any domain in the forest. If Jim, in the sales.cohovineyard.com domain, needed access to a printer in the cs.cohowinery.com domain, Tom could grant this access. Because in this example cs.cohowinery.com is the trusting domain and sales.cohovineyard.com is the trusted domain, Sarah, a domain administrator in the cs.cohowinery.com domain, also could grant permission to use the printer. Bob, a domain administrator for sales.cohovineyard.com, could not grant such permissions, however, because the printer resource exists in a domain other than the one he controls.

To continue working with Figure 10-10, take a look at the arrows that designate the trust relationships. For a user in the sales.cohovineyard.com domain to access a printer in the cs.cohowinery.com domain, the request must pass through the following series of trust relationships:

1. The trust between sales.cohovineyard.com and cohovineyard.com
2. The trust between cohovineyard.com and cohowinery.com
3. The trust between cohowinery.com and cs.cohowinery.com

The *trust path* defines the path that an authentication request must take between the two domains. Here, a domain controller in the user's local domain (sales.cohovineyard.com) would pass the request to a domain controller in the cohovineyard.com domain. This domain controller, in turn, would pass the request to a domain controller in the cohowinery.com domain.

Finally, the request would be passed to a domain controller in the cs.cohowinery.com domain, which would ultimately grant or deny access.

In all, the user's request has to pass through four domain controllers—one for each domain between the user and the resource. Because the domain structure is separate from the network's physical structure, the printer could actually be located right beside the user's desk and the user would still have to go through this process. If you expand this scenario to include all the users in the sales.cohovineyard.com domain, you could potentially have hundreds of users whose requests have to go through a similar process to access resources in the cs.cohowinery.com domain.

Omitting the fact that the domain design in this scenario is very poor—because if many users are working with resources, those resources are ideally in their own domain or in a domain closer in the tree—one solution for this problem would be to establish a *shortcut trust* between the user's domain and the resource's domain. With a shortcut trust, you could specify that cs.cohowinery.com explicitly trusts sales.cohovineyard.com. Now when a user in the sales.cohovineyard.com domain requests a resource in the cs.cohowinery.com domain, the local domain controller knows about cs.cohowinery.com and can directly submit the request for authentication. This means that the sales.cohovineyard.com domain controller sends the request directly to a cs.cohowinery.com domain controller.

Shortcut trusts are designed to help make more efficient use of resources on a busy network. On a network with a lot of activity, the explicit trust can reduce the overhead on servers and on the network as a whole. You shouldn't implement shortcut trusts without careful planning. You should use them only when resources in one domain will be regularly accessed by users in another domain. They don't need to be used between two domains that have a parent–child relationship because a default trust already exists explicitly between a parent domain and a child domain.

With Active Directory, you can also make use of *external trusts*. External trusts are manually configured and are always nontransitive. External trusts can be either one-way or two-way trusts. When you establish a trust between a domain in one forest and a domain in another forest, security principals from the external domain can access resources in the internal domain. In the internal domain, Active Directory creates a foreign security principal to represent each security principal in the external domain. Foreign security principals can be added to domain local groups in the internal domain.

Active Directory namespaces and partitions

Any data stored in the Active Directory database is represented logically as an object. Every object in the directory has a relative distinguished name (RDN). That is, every object has a name relative to the parent container in which it's stored. The relative name is the name of the object itself, and it's also referred to as an object's *common name* (CN). This relative name is stored as an attribute of the object and must be unique for the container in which it's located. Following this, no two objects in a container can have the same common name, but two objects in different containers could have the same name.

In addition to an RDN, objects have a distinguished name (DN). An object's DN describes the object's place in the directory tree and is logically the series of containers from the highest to the lowest of which the object is a part. It's called a distinguished name because it serves to distinguish like-named objects and, as such, must be unique in the directory. No two objects in the directory will have the same distinguished name.

Every object in the directory has a parent, except the root of the directory tree, which is referred to as the rootDSE. The rootDSE represents the top of the logical namespace for a directory. It has no name per se. Although there is only one rootDSE, the information stored in the rootDSE specifically relates to the domain controller on which the directory is stored. In a domain with multiple domain controllers, the rootDSE will have a slightly different representation on each domain controller. The representation relates to the capability and configuration of the domain controller in question. In this way, Active Directory clients can determine the capabilities and configuration of a particular domain controller.

Below the rootDSE, every directory tree has a root domain. The root domain is the first domain created in an Active Directory forest and is also referred to as the forest root domain. After it's established, the forest root domain never changes, even if you add new trees to the forest. The LDAP distinguished name of the forest root domain is DC=*ForestRootDomainName*, where DC is an LDAP identifier for a domain component and *ForestRootDomainName* is the actual name of the forest root domain. Each level within the domain tree is broken out as a separate domain component. For example, if the forest root domain is cohovineyard.com, the domain's distinguished name is DC=cohovineyard,DC=com.

When Active Directory is installed on the first domain controller in a new forest, three containers are created below the rootDSE:

- The Forest Root Domain container, which is the container for the objects in the forest root domain
- The Configuration container, which is the container for the default configuration and all policy information
- The Schema container, which is the container for all objects, classes, attributes, and syntaxes

From a logical perspective, these containers are organized as shown in Figure 10-11. The LDAP identifier for an object's common name is CN. The DN for the Configuration container is CN=configuration,DC=*ForestRootDomainName*, and the DN for the Schema container is CN=schema,CN=configuration,DC=*ForestRootDomainName*. In the cohovineyard.com domain, the DNs for the Configuration and Schema containers are CN=configuration,DC=cohovineyard,DC=com and CN=schema,CN=configuration,DC=cohovineyard,DC=com, respectively. As you can see, the distinguished name allows you to walk the directory tree from the relative name of the object you are working with to the forest root.

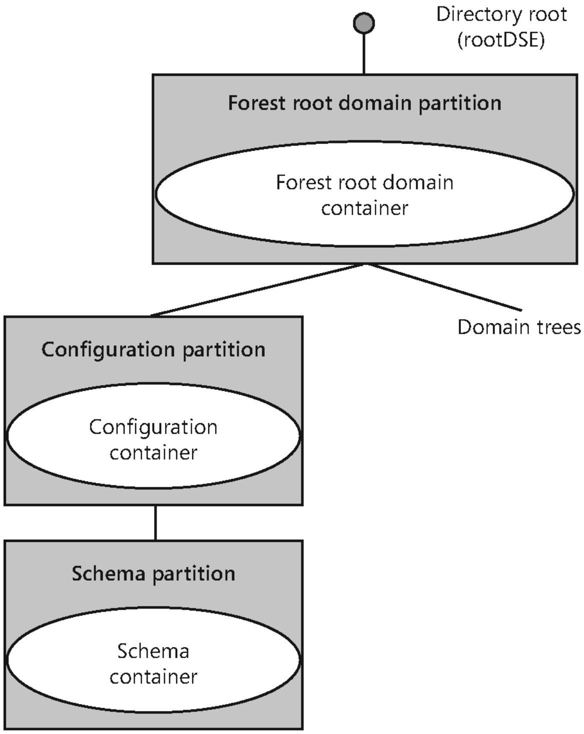

Figure 10-11 The directory tree in a new forest.

As shown in the figure, the Forest Root Domain container and the Configuration and Schema containers exist within their own individual partitions. Active Directory uses partitions to logically apportion the directory so that each domain controller does not have to store a complete copy of the entire directory. To do this, object names are used to group objects into logical categories so that the objects can be managed and replicated as appropriate. The largest logical category is a directory partition. All directory partitions are created as instances of the domainDNS object class.

As far as Active Directory is concerned, a domain is a container of objects that is logically partitioned from other container objects. When you create a new domain in Active Directory, you create a new container object in the directory tree, and that container, in turn, is contained by a domain directory partition for the purposes of management and replication.

Active Directory data distribution

Active Directory uses partitions to help distribute three general types of data:

- Domainwide data, which is data replicated to every domain controller in a domain
- Forestwide data, which is data replicated to every domain controller in a forest
- Application data, which is data replicated to an arbitrary set of domain controllers

Every domain controller stores at least one domain directory partition and two forestwide data partitions: the schema partition and the configuration partition. Data in a domain directory partition is replicated to every domain controller in the domain as a writeable replica.

Forestwide data partitions are replicated to every domain controller in the forest. The configuration partition is replicated as a writeable replica. The schema partition is replicated as a read-only replica, and the only writeable replica is stored on a domain controller that is designated as having the schema operations master role. Other operations master roles also are defined.

Active Directory can replicate application-specific data that is stored in an application partition, such as the default application partitions used with zones in Domain Name System (DNS) that are integrated with Active Directory. Application partition data is replicated on a forest-wide, domainwide, or other basis to domain controllers that have a particular application partition. If a domain controller doesn't have an application partition, it doesn't receive a replica of the application partition.

In addition to full replicas that are distributed for domains, Active Directory distributes partial replicas of every domain in the forest to special domain controllers designated as global catalog servers. The partial replicas stored on global catalog servers contain information on every object in the forest and are used to facilitate searches and queries for objects in the forest. Because only a subset of an object's attributes is stored, the amount of data replicated to and maintained by a global catalog server is significantly smaller than the total size of all object data stored in all the domains in the forest.

Every domain must have at least one global catalog server. By default, the first domain controller installed in a domain is set as that domain's global catalog server. You can change the global catalog server, and you can designate additional servers as global catalog servers as necessary.

Index

M

N

O

P

Q

R

S

U

V

W

X

Z